Master/Worker Parallel Discrete Event Simulation

A Thesis
Presented to
The Academic Faculty

by

Alfred J. Park

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
May 2009

Master/Worker Parallel Discrete Event Simulation

Approved by:

Dr. Richard Fujimoto, Advisor
Computational Science and Engineering Division
Georgia Institute of Technology

Dr. George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David Bader
Computational Science and Engineering Division
Georgia Institute of Technology

Dr. Richard Vuduc
Computational Science and Engineering Division
Georgia Institute of Technology

Dr. Kalyan Perumalla
Computational Sciences and Engineering Division
Oak Ridge National Laboratory

Date Approved: December 1, 2008

ACKNOWLEDGEMENTS

First and foremost, this thesis would not have been possible without the support and guidance from my advisor, Dr. Richard Fujimoto. It has truly been an honor and a blessing to perform research under the supervision of one of the pioneers in the parallel and distributed simulation field. I feel I have learned a great deal not only from his advisement and expertise but also through the exposure to a variety of projects that I was able to participate in. I am extremely grateful to have had the opportunity to study as one of his students.

I would like to express my gratitude to Dr. George Riley and Dr. Kalyan Perumalla who have been very supportive of my work. Dr. Riley has provided me with unique challenges that allowed me to broaden my horizons from early work with Parallel and Distributed NS-2 to collaborative work with GTRI. I was fortunate enough to work with Dr. Perumalla at ORNL with the SensorNet Group, and for that I am extremely grateful. I would like to also thank Dr. David Bader and Dr. Richard Vuduc for their unique perspective and input to help better the quality of the thesis.

I would like to thank people who I was lucky enough to collaborate or interact with: Dr. Adelinde Uhrmacher, Matthias Jeschke, Roland Ewald, Dr. Michael Hunter, Dr. Andy Register, Dr. Dale Blair, and Dr. David Sherrill. I would like to also thank other graduate students I interacted with: Steve Ferenci, Jen-Chih Huang, Yan Gu, Eric Liu, Jagrut Dave, Daniele Gianni, Hao Wu, Mahesh Palekar, and Chris Thompson. I appreciate all of the help and assistance given to me by Lometa Mitchell and Carolyn Young.

Finally, I would like to express my sincere gratitude to my parents and my brother for their loving support through the years. Without them, this work would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Recent advances in metacomputing such as volunteer and desktop grid computing that aggregate loosely coupled resources have transformed the execution of certain computational workloads that, in the past, were reserved for processing on dedicated clusters. Parallel discrete event simulations have different requirements than programs that can readily exploit loosely coupled resources such as embarrassingly parallel codes. Consequently, parallel discrete event simulations are typically run on tightly coupled machines providing the best opportunity for maximum speedup. However, these facilities may not be readily available to many users.

The focus of this thesis explores the merging of these distinct computational domains involving the execution of parallel discrete event simulation across loosely coupled resources. A master/worker architecture for parallel discrete event simulation is proposed providing robust executions under a dynamic set of services with system-level support for fault tolerance, semi-automated client-directed load balancing, portability across heterogeneous machines, and the ability to run codes on idle or time-sharing clients without significant interaction by users. Results indicate that a master/worker approach utilizing loosely coupled resources is a viable means for high throughput parallel discrete event simulation by enhancing existing computational capacity or providing alternate execution capability for less time-critical codes.

Research questions and challenges associated with issues and limitations with the work distribution paradigm, targeted computational domain, performance metrics, and the intended class of applications to be used in this context are analyzed and

discussed. A portable web services approach to master/worker parallel discrete event simulation is proposed and evaluated. Optimizations to increase the efficiency of large-scale simulation execution through distributed master service design and intrinsic overhead reduction are proposed and evaluated. Finally, challenges for optimistic parallel discrete event simulation such as rollbacks and message unsending with an inherently different computation paradigm utilizing master services and time windows are addressed and evaluated.

CHAPTER 1

INTRODUCTION

1.1 Background

Computer simulation is a means to model real-world systems and processes under a controlled environment with access to variables that can modify results and outcomes [1]. Simulation is a widely-used, invaluable tool for development, research and testing in many scientific and engineering domains. For example, computer processor engineering teams may utilize simulators to test effectiveness and efficiency of certain cache sizes, associativity, eviction and replacement policies. By utilizing simulations, the cost and burden of creating hardware prototypes in the early stages of development can be avoided. Computer network simulations are often used to model protocol behavior and performance of wired networks as well as development and testing of new wireless and sensor network mechanisms. Simulations can be used to model virus, worm, and distributed denial of service attacks to better understand large-scale behavior in order to evaluate counter-measures and enable research of security protocols against these threats.

Simulations in the physical sciences provide tremendous benefits by enabling what-if or virtual experiments in lieu of performing them physically in a laboratory. For instance, biological simulations are becoming increasingly important and relevant with the increased computational processing power available today. Simulations of protein folding and misfolding behavior help scientists understand the origins of certain diseases and possible approaches to cures. Recently, major natural disasters such as hurricanes linked with climate change and global warming are placing an increasing importance on environmental and atmospheric simulations. Simulations are used to estimate the number

of hurricanes expected during a season as well as forecasting hurricane trajectories to provide early warning to initiate life-saving evacuations. These simulations in turn can be used with other simulation models such as traffic simulators to help expedite evacuation plans on a city or regional scale. Simulations are often used in the military as a testing and evaluation alternative to live exercises as well as to help plan proper courses of action. These simulations are invaluable as cost saving measures and also can aid commanders with additional information so they can make the best possible decisions to manage resources under their control.

Computer simulations can be generally divided into two categories: continuous and discrete. As the name implies, continuous simulations correspond to models that change continuously over time. These continuous simulations are often modeled numerically as sets of differential equations providing precise calculation of state rate changes with respect to time. Continuous simulations can be formulated for many problems where the exact mathematical behavior of the system or behaviors of the components that make up the system are known. For example, continuous simulation models of digital circuits is possible as mathematical models for components such as transistors and capacitors are precisely known. Due to the amount of computation involved in continuous models, these simulations can exhibit poor performance when modeling non-trivial problems; for example, digital circuits of complex computer components that contain millions of transistors may be too computationally expensive to perform.

Discrete simulations model physical systems by assuming changes in state occur at distinct points in time. These discretized changes to the system state are ordered as

events, with each event assigned a timestamp. Events may be pre-generated such as airline departure times in an airport simulation, or randomly generated such as packets generated in a sensor network. These events, when processed, may change the state of the system and generate new events to be processed later. Discrete models can be further categorized by how they advance in time, either time-stepped or event-stepped.

Discrete simulations where advancement in simulation time is through fixed time intervals are referred to as time-stepped simulations as shown in Figure 1. State variables are computed, if necessary, during each fixed time-step interval.

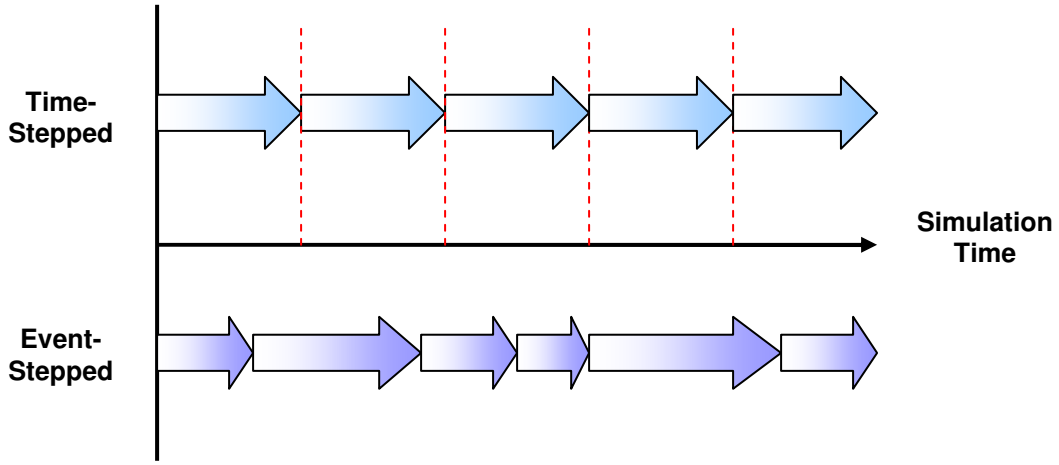

Figure 1: Discrete Simulation Types

Sometimes no state variables need to be updated during time steps; an alternate method to discrete simulation time advancement is event-stepped execution. Discrete simulations that are event-stepped eliminate the need for fixed intervals by allowing simulations to perform jumps in time advancing from one event to the next in the event queue in time stamped order (TSO). Unprocessed events are stored in a data structure called the event queue, and are removed in non-decreasing timestamp order for processing. In addition to the event queue, a collection of variables are defined that

represent the state of the system being modeled. Discrete event simulations implement a clock representing the current simulation time. Simulation time is a time scale that represents time in the system being modeled. In general, simulation time advances independent of wall clock time. A discrete event simulation with a simulation time clock value of *n* indicates that all events with a timestamp prior to time *n* have been processed. Moreover, state variables that may have been modified as the result of processing events have been updated.

This thesis is primarily concerned with discrete event models and in particular, parallel discrete event simulations which are described next.

1.1.1 Parallel Discrete Event Simulation

A simulation is an approximation of reality. Deficiencies in model fidelity, scale or perhaps even insulation of the model from external factors (e.g., simulation models that do not take into account interaction effects from other sources, for example a sensor network simulator measuring environmental conditions without an accurate environmental simulator and model [2]) can prevent simulation results from matching reality. These limitations sometimes result from a lack of understanding and consequently insufficient detail in the model. Often approximations are made to trade-off between accuracy and computational requirements. For instance, a network simulation executing on a single machine is limited by the amount of memory available for storing relevant data such as routing tables; either the scale of the simulation is limited or less accurate techniques are used to save computation time or space. Parallel and distributed simulation addresses these limitations by allowing the model to be spread across many

processors. This can enable enhanced model fidelity, larger model scale, integration with other simulators and/or reduced execution time.

A discrete event simulation that is partitioned and executes across more than one processor is referred to as a parallel discrete event simulation (PDES). PDES enables the execution of simulations such as computer network models that exceed the capability of a single machine. In addition, PDES is also used to speed up the execution of a sequential execution. Larger and faster network simulations allow computer network researchers to examine wide-area end-to-end performance and behavior across very large networks containing millions of simulated nodes [3]. Traffic simulations whether they are used to model vehicular ground traffic, rail systems, or air traffic are widely used for planning, management, and what-if simulation of emergency scenarios. Similarly, PDES has been applied to biological and environmental simulations and military wargaming. PDES extends of the applicability of sequential discrete event simulation to allow larger-scale models, higher fidelity and faster return of results than would otherwise be possible.

In order to execute a discrete event model across many processors as a PDES, the model must be partitioned into segments. A priori spatial partitioning of the model is often used where the problem is decomposed into logical processes (LPs) where each LP represents some fraction of the entire model. Suppose the model of interest is an air traffic control PDES simulation covering the entire United States of America. A sample spatial partitioning scheme is shown in Figure 2 where the NW portion of the USA is mapped to LP 1, NE portion to LP 2, SW portion to LP 3, and the SE portion to LP 4.

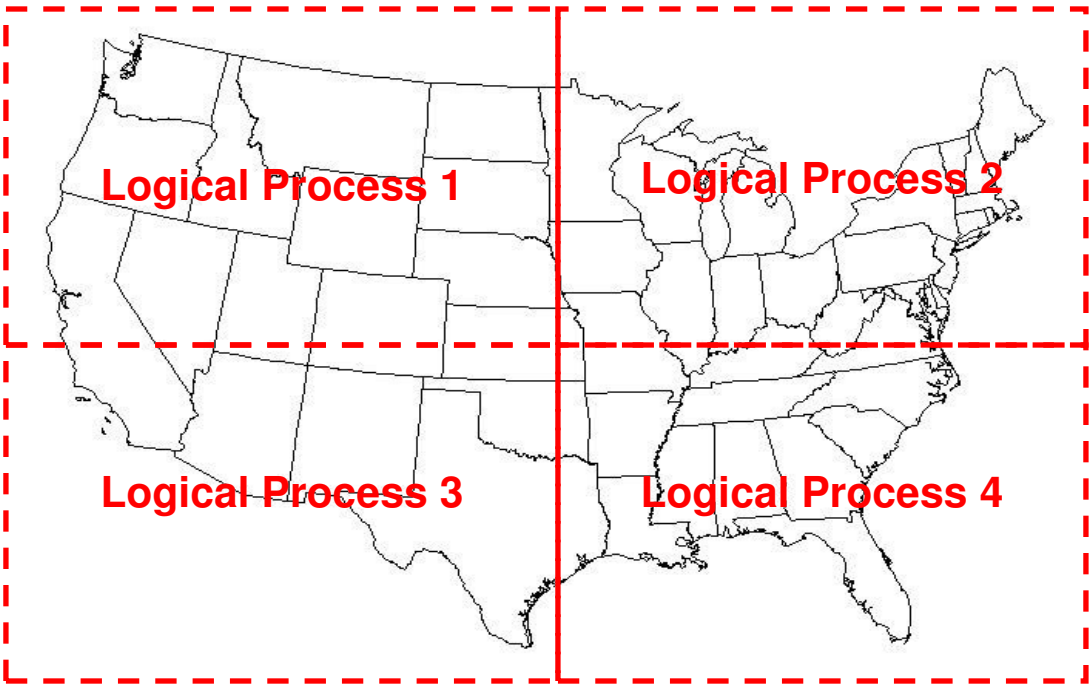

Figure 2: Spatial Partitioning

Communication between LPs occurs through message passing. Inter-LP events that originate from a source LP that differs from the destination LP are sent as messages that have associated simulation timestamps for when they should be processed. For this example, aircraft may pass between any LP, thus, messages can be sent between any LP. Since messages can be sent from and received by any LP in the system, this model exhibits a fully-connected LP topology. LP topologies are dependent upon the problem and the partitioning the modeler chooses if performed manually. If, in the entire model across all LPs, no aircraft moved between LPs, then there would be no messages exchanged between LPs. Each LP could run unrestricted and process all events to simulate the entire desired simulation time period as there would be no interaction between LPs. However, most non-trivial PDES models will often generate events as

messages which cross inter-LP boundaries. For example, in this air traffic control scenario, an aircraft departing from Atlanta, Georgia destined for Seattle, Washington must cross one or more LP boundaries. This can be represented as a departure event in LP 4 with an arrival timestamp sometime later at LP 1. Different airline routes may have the aircraft passing through LP 2 on its way to LP 1, which may model the event first "arriving" in LP 2's airspace, followed by a later departure and entry into LP 1's airspace. Thus, a new problem is introduced where the distributed simulation must now synchronize LPs so that these events can be correctly processed without a causality violation.

Out-of-order event execution must be prevented to ensure correct execution. This synchronization problem in PDES calls for time management schemes which operate on a principle called the Local Causality Constraint (LCC). The LCC states that each LP must process events in non-decreasing TSO. The preservation of LCC guarantees that a PDES execution will produce the exact same results as a sequential execution with the assumption that simultaneous events are processed in the same order for both parallel and serial versions.

From the previous example of an aircraft departing from Atlanta, Georgia at time T and destined for Seattle, Washington at time T + 5, the PDES must prevent events in LP 1 from processing events after T + 5 before the receipt of the arrival event. If events are processed after T + 5, the simulation time of LP 1 will be in the future while a message notifying that an aircraft will be arriving is in the simulated past. Adhering to the LCC through a synchronization protocol would prevent LP 1 from executing events which are too far into the future and are unsafe for processing. The class of

synchronization algorithms that prevent LCC violations are collectively referred to as conservative time management. Mechanisms which allow violations of the LCC to occur but perform operations to correct causality errors are known as optimistic time management. Each of these synchronization schemes will be discussed in detail later. All PDES simulations must incorporate synchronization techniques to reproducibly execute a discrete event simulation across many processors to free the simulation from limitations of a serial execution providing benefits mentioned earlier.

As with most parallel computations, the proportion of the program that can be parallelized must be relatively large relative to the portion that is inherently serial.

$$f(N) = \frac{1}{T_s + (1 - T_s)/N} \tag{1.1}$$

Amdahl's law is expressed in equation (1.1). Here, *f(N)* represents the maximum speedup that can be achieved using N processors. T_S denotes the fraction of the computation that is inherently sequential. Even with an arbitrarily large number of processors, the speedup can be no larger than the inverse of the portion of the program that is inherently sequential. This suggests that even with small amounts of serial computation, the overall speedup that can be achieved from parallelization can be severely limited. Thus one requirement for PDES codes to achieve effective parallelization is the fraction of the code that is inherently serial must be small.

A second requirement concerns the amount of computation that takes place between interprocessor communications. Discrete event simulations that exhibit low a high computation to communication ratio, i.e., a large amount of computation between communications, are well-suited for parallelization. Fine-grained simulations are those

that only process a small number of events before communicating with another processor. The computation to communication ratio can be increased by mapping many LPs that communicate amongst themselves to the same processor. This thesis is targeted toward coarse-grained PDES simulations with a substantial amount of computation between communications. One element of the research described in this thesis is to quantify this aspect.

1.1.1.1 Conservative Synchronization

Conservative synchronization protocols are the class of mechanisms that do not allow any violations in ordered event execution to occur. Thus all events are processed in TSO, within each LP. Out-of-order execution of events must be avoided in order to preserve the LCC. Conservative synchronization mechanisms in PDES rely on a value called lookahead. Lookahead is defined as the minimum time value that must be added to an event relative to the current simulation time of the LP when the event is generated. For example, assume that an air traffic PDES simulation contains two LPs. LP A represents the airport in Atlanta, while LP B represents an airport in Seattle. Air travel between the two airports requires a minimum amount of time dependent upon the maximum velocity of an aircraft. Suppose that it takes a minimum of 5 hours for an airplane to fly from Atlanta to Seattle. This means that no matter what the circumstances, an airplane departing from Atlanta at simulation time T cannot arrive before T + 5. This value of 5 hours is the minimum amount of time into the future that the departure event can schedule an arrival event for the Seattle LP. This minimum amount of time is lookahead. This guarantee increases the amount of concurrent execution that can occur in a PDES execution.

Virtually all conservative mechanisms use lookahead values to calculate a value known as the Lower Bound Time Stamp (LBTS) that is for synchronization purposes. $LBTS_i$ is defined as the smallest timestamp of any event can be delivered to LP_i in the future among adjacent LPs, i.e., those LPs that are able to send events to LP_i. These $LBTS_i$ values are used to synchronize the simulation so that no LP is able to process an event if it is possible a smaller time stamped event might later arrive. A sample asynchronous conservative time management scheme utilizing LBTS values is illustrated in Figure 3.

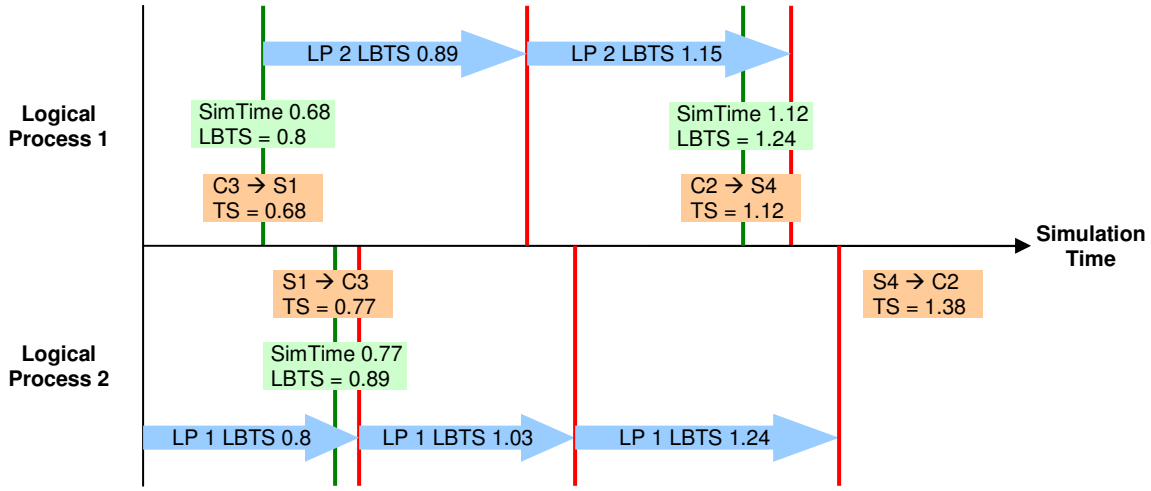

Figure 3: Event Processing and Asynchronous Conservative Time Advancement

In Figure 3, the orange boxes denote events while the green boxes contain the LP simulation time and calculated LBTS values. At the beginning of the simulation, both LPs advance to the first available event in their local event queues. Here LP 1 has an event at time 0.68. This event is processed since it is the smallest time stamped event in the system and LP 2 cannot send an event with a smaller timestamp. Once processed, LP 1 calculates an $LBTS_1$ value of 0.8 that is obtained by adding the current simulation time

0.68 to the lookahead value of 0.12. LP 2 can advance to this time and is able to process the event with timestamp 0.77. $LBTS_2$ is now computed to be 0.89. When LP 1 attempts to process the event with timestamp 1.12 it cannot proceed because $LBTS_2$ is 0.89. Thus LP 1 must block at this simulation time until a later time guarantee. Next, LP 2 attempts to process the event at 1.38. This event is not safe to process yet because $LBTS_1$ is 1.03. LP 1 is now able to process the event with timestamp 1.12, because the smallest timestamp event that can later be received in now 1.15.

Of particular interest with regard to the work presented here are synchronous conservative synchronization methods involving simulation time windows. One such method is the algorithm used in the Yet Another Windowing Network Simulator (YAWNS) [4] and similarly the time bucket synchronization mechanism used in the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) framework [5]. The minimum time stamped message among all LPs is found along with its associated lookahead value. The window of simulation time available for execution is simply the current simulation time of the LP and this calculated global minimum time stamped message plus the lookahead value. Any events falling within this window can be safely processed.

Several other conservative protocols have been proposed. Null messages [6, 7] is an asynchronous "local" synchronization scheme where "null" messages from a LP are sent to all neighboring processes indicating a lower bounds on any future message that may be sent. These guarantees are used by LPs to safely process events. The null messages are also used to avoid deadlock situations that can arise when a cycle of LPs forms where each LP is waiting for the next LP in the cycle. A simulation that exhibits

small lookahead values can often cause poor performance because an excessive number of null messages can be generated.

In contrast to null messages that actively avoid deadlocks, deadlock detection and recovery [8, 9] is another asynchronous conservative synchronization scheme that allows deadlocks to occur but mechanisms are used to detect and break deadlocks to keep the simulation advancing forward. The concept of diffusing computations can be used where the receipt of a message triggers computation and the possible generation of new messages. A tree data structure is used to detect deadlock. The recovery phase involves identifying events that are safe to process and preserve the LCC. These LPs are signaled to process safe events and the computation resumes until the next deadlock. The process of deadlock-detection-recovery is repeated until the simulation completes.

There exist synchronous synchronization methods that utilize barrier algorithms [10]. Transient messages (e.g., messages which are delayed in the network) must be taken into account when using barrier synchronization methods to avoid incorrect execution. A solution to the transient message problem with barrier algorithms is to use send and receive message counters. After a global barrier for synchronization, the system can examine the aggregate total between the send and receive counters. If they are equivalent, then no transient messages exist and guarantees can be made. Centralized barriers using controller processes are simple to implement but scale poorly as the number of processors increase. Other barrier mechanisms such as the butterfly barrier reduce the amount of messages required for synchronization over centralized schemes.

Distance between objects [11] is a mechanism to determine the minimum "distance" between LPs that may not be adjacent but can affect each other through a path

of links from one LP to another. By calculating the minimum LBTS values of all possible LPs that can affect the LP for whom the LBTS value is being computed, the simulation can potentially process more events simultaneously than a conservative synchronization mechanism that only takes into account directly adjacent LPs.

Bounded lag [12] uses the distance between objects idea but differentiates between LPs that can affect the LP for which the LBTS value is being computed and those which cannot. Events which fall outside of the current simulation time plus a time window need not be checked and are never processed during the current execution window. More precisely this means that if the lookahead between LP A to LP B exceeds the time window, then events generated from LP A to LP B need not be checked for synchronization purposes. This reduces the amount inter-processor communication for synchronization. A problem, as with many schemes that employ time windows, is the question of how to set the window size.

1.1.1.2 Optimistic Synchronization

Optimistic synchronization algorithms relax the requirements for strictly adhering to the LCC [13]. Rather, incorrect executions can occur where events may be delivered in the simulated past of an LP (referred to as a straggler message). Time Warp is the most well-known optimistic synchronization algorithm. In Time Warp, a straggler message is a message sent from a source LP to a destination LP where the receive timestamp is less than the current simulation time of the destination LP. Additional mechanisms are required to detect, correct, and restart the execution. When an event is delivered in the past of a logical process, a rollback phase is initiated where the state of the logical process is restored to a simulation time preceding that of the straggler message. Messages

sent by rolled back computations are “unsent” using anti-messages. This process is shown in Figure 4.

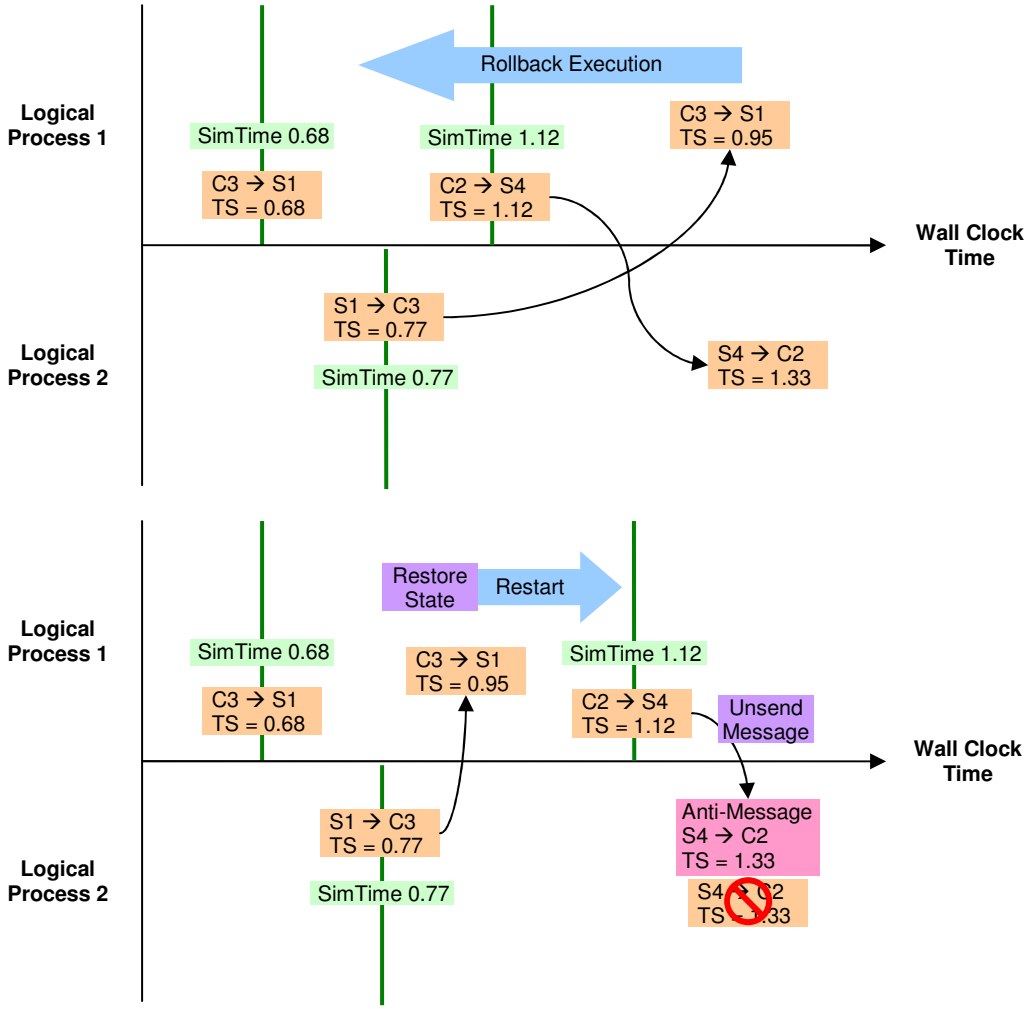

Figure 4: Handling a Straggler Message via Rollback

Under an optimistic execution, an LP may encounter events in the event queue that fall in the LP’s simulated past. Suppose the event at time 0.77 in LP 2 generates an event for LP 1 at 0.95, but LP 1 has already processed an event at time 1.12. This straggler message triggers a rollback in LP 1. First, the state of the LP is restored to that that existed at simulation time 0.95. If no state at the exact rollback time exists, the most

recent state prior to the rollback time is used. In the latter scenario, coast forwarding is enabled, where recomputation of the simulation occurs until the rollback time is reached. Coast forwarding is a mechanism to ensure correct recovery after a rollback if no state exists at the rollback time. During the coast forwarding phase, both positive and anti-message sending is disabled to prevent duplicate messages from being generated. Once the rollback time is reached, coast forwarding is disabled and normal recovery is resumed. Since the processed event at time 1.12 at LP 1 also sent an event to LP 2, this message must be unsent. An anti-message is generated and sent to LP 2. When LP 2 receives this anti-message, it annihilates the "positive" message in the event queue. In the case where the positive message has already been processed, a rollback must be performed to remove the erroneous event computation. This is known as a secondary rollback. After all anti-messages have been sent, the simulation is then restarted from the rollback point, processing the new message at 0.95.

As optimistic simulations progress forward, more and more memory is consumed for state saving and anti-messages. If memory is not released, the simulation may run out of memory and fail to complete. Not all memory can be freed readily, and the issue of what memory is safe to release along with committing irrevocable actions such as I/O is solved through the global control mechanism. The Global Virtual Time (GVT) value is similar to the LBTS value used in conservative synchronizations. It is a system-wide minimum timestamp of any future rollback that can occur. GVT is defined as the minimum timestamp among all unprocessed messages in the system, including partially processed messages and anti-messages. Computation of the GVT value allows saved state and unsent anti-messages to be safely released as the simulation proceeds. GVT

computations are usually computed asynchronously, e.g., using Samadi's [14] or Mattern's [15] algorithm.

There has been much work detailing optimizations to state saving, as this can be a major source of overhead. Copy state saving is the most basic mechanism. This method simply makes a copy of all of the current state variables of an LP before an event is processed. This can lead to high memory consumption and overhead in fine-grained executions. Infrequent state saving is a technique to only save state prior to every *n*th event [16]. If a rollback occurs and no proper state can be found, coast forwarding as described earlier is used to re-compute the proper state before restarting the simulation at the rollback time. Incremental state saving is a mechanism to store only state variables that have been modified since the last save [17-19]. Simulations that exhibit only a small subset of variables that are modified from event to event can benefit from memory savings and overhead using incremental state saving. Simulations that modify most of the state variables during event processing are better suited for copy state saving techniques that are not burdened by saving memory addresses of variables that have been modified.

A well-known problem with optimistic protocols is the need to limit the amount of rolled back computation. One approach is to provide a bound on how far into the future each LP can process events. Optimism control schemes operate on the premise that by preventing LPs from executing too far into the future (e.g., further away from GVT), the probability of errant computations and invalid messages generated will be less, thus reducing the number of rollbacks. By limiting optimism, the number of incorrect computations can be potentially reduced. The Moving Time Window (MTW) protocol

sets execution limits on how far into the future an LP may progress by defining a time window, and preventing LPs from executing beyond the upper edge of the window [20]. The window length is usually specified by the modeler or adaptively tuned as the simulation runs; this window amount is added to the current GVT value to give the total available amount of simulation time given to an LP to process. There are several issues with this approach. The first concerns how to set the size of the window. The stochastic processes determining event generation in some simulations can be unpredictable. Secondly, even with an optimal window size, there is no restriction within the actual window to prevent incorrect event computation and thus rollbacks, although restricted through limited execution via windowing, may still occur.

Another scheme for optimism control is SRADS [21] and the Breathing Time Buckets (BTB) protocol [5]. Any message that is generated is not immediately sent, but rather buffered locally. Next, the minimum of all the receive timestamps recorded by all LPs is computed, known as the event horizon. Each LP will process events with timestamps less than their local event horizon. The global event horizon representing the minimum event horizon value across all local event horizons is computed. Once this value has been computed and made known to all LPs, buffered messages that have a send timestamp earlier than the global event horizon can be sent. These messages are guaranteed not to rollback, and the GVT value can be set to the computed global event horizon time. The SRADS and BTB protocols are considered risk-free optimistic approaches in that they eliminate the need for anti-messages and thus avoid secondary rollbacks, i.e., rollbacks caused by anti-messages. The problem with this approach is that it may be overly conservative and the number of events that can be processed within each

execution cycle may be insufficient to provide enough concurrent computation for significant speedup. Breathing Time Warp (BTW) [22] attempts to combine BTB and Time Warp by allowing events close to GVT to be executed under a Time Warp protocol while events that are far away from GVT execute under the BTB protocol. A problem with this approach is effectively determining the proper time boundaries for when the Time Warp phase should end and the BTB phase should begin for each execution cycle.

The main optimistic processes described such as rollback, recovery, and anti-messages along with other issues such as handling program errors were implemented in the Time Warp Operating System (TWOS) [13]. Other optimizations to the original Time Warp mechanism have been proposed. Direct message cancellation [23, 24] exploits shared memory architectures by using pointers to implement anti-messages. One of the issues with Time Warp is the possibility the simulation may run out of memory due to saving stating and anti-messages. The pruneback protocol [25] provides a means to reclaim memory by selectively pruning copy save state vectors. Cancelback [26, 27] is a technique proposed using message sendback [13] as a means to reclaim memory by returning messages back to their sender. Artificial rollback [28] is another technique to reclaim memory similar to cancelback, but instead utilizes the rollback protocol to reclaim memory. An alternate non-reactive strategy to address memory issues with Time Warp is to pro-actively attempt to prevent memory exhaustion through blocking [29]. Systems employing this strategy only provide blocks of memory in cycles between fossil collections adaptively allocating memory based on predictions on memory usage. If the allocated memory is exhausted by the LP, the LP blocks until memory is reclaimed. These memory reclamation techniques can be considered as optimism control

mechanisms as they limit how far a LP can process indirectly through memory recovery. Other mechanisms have been proposed that directly limit optimism through other means such as probabilistic adaptive direct optimism control [30] that analyzes message arrival histories of the simulation to adaptively set the synchronization method to a blocking-based conservative mechanism or an optimistic Time Warp protocol. Reverse computation [31] is an approach to avoid overheads associated with state saving. When a rollback occurs, instead of restoring a known valid saved state, the PDES program processes the inverse code from the current simulation time back to the rollback time. Once the program has run "backwards" to the rollback time, the correct simulation state has been restored without the overhead of state saving and can begin forward simulation progress.

1.1.2 Embarrassingly Parallel Codes

In contrast to PDES, many parallel and distributed programs involve problems that fall into the embarrassingly parallel (EP) class of computational codes. EP programs are typically classified by their lack of dependency between partitions of work (e.g., non-existent interprocess communication) and trivial partitioning of the problem into parts. Given sufficient resources, these programs offer nearly linear speedups because of these advantages.

$$\lim_{N \to \infty} f(N) = T_s^{-1} \qquad (1.2)$$

When an unlimited number of processors are available, Amdahl's law can be expressed as the inverse of the non-parallelizable portion of a parallel and distributed program as shown in equation (1.2). EP codes typically have non-existent interprocess or

inter-partition communication thus the value of T_S is close to zero yielding very large values for *f(N)*.

An example EP application is a simulation based on the Monte Carlo method. A simple Monte Carlo method utilizes random numbers generated from probability distributions that closely resembles existing data that are processed through a mathematical model. Results are then aggregated to produce a final answer. For instance, the following Monte Carlo problem attempts to estimate π given the volume of a specific sphere with a radius of 2 centered at the origin:

$$x^2 + y^2 + z^2 = 4 \tag{1.3}$$

A computer program can generate uniformly random values for x, y, and z. These three random values can be evaluated using equation (1.3). If the three-dimensional point falls within the sphere space, the result of the Monte Carlo trial is considered a "hit."

$$\frac{\frac{4}{3}\pi r^3}{(r \times r)^3} \approx \frac{\#\ hits}{\#\ valid\ trials} \tag{1.4}$$

$$\pi \approx \frac{6 \times \#\ hits}{\#\ valid\ trials} \tag{1.5}$$

Equation (1.4) represents the proportional relationship between known equations of the volume of a sphere and the volume of cube to the Monte Carlo hits and Monte Carlo trials that fall within the volume of cube. Solving for π yields equation (1.5).

Monte Carlo methods such as these hit-and-miss scenarios are embarrassingly parallel as trials can be computed independently using any number of threads or processors without the need for synchronization. To produce an answer, all that is required is an aggregation of results at the end and the evaluation of equation (1.5). Lack

of dependency and communication during computation allow EP codes such as Monte Carlo simulations to be highly scalable and massively distributed in nature.

Task parallelism involves programs that are run in parallel to produce multiple replications or achieve large speedups over sequential executions that are repeated one after another. At the most fundamental level, these codes can be classified as an EP style of work distribution and computation. Task parallelism can be fine- or coarse-grained. Fine-grained task parallelism consists of distributing a portion of the program to other available processors such as parallelizing the execution of data-independent for-loops in programs like that of a matrix multiplication problem. Several libraries exist to ease implementation on the programming language level such as OpenMP [32], Microsoft's .Net Task Parallel Library (TPL) [33] and the MATLAB Parallel Computing Toolbox [34]. Coarse-grained task parallel programs are those where large portions or complete programs are tasked to processors independently. For instance, in modeling hurricane predictions, a set of variables that affect hurricane trajectory are tested. For each set of modifications to environmental variables, tasks can be created and run independently on separate machines. This allows the gathering of different trajectory data in parallel without having to run each task one after the other. The limiting factors to this approach are the amount of computational resources available and per-node memory and disk capacities. The following is a brief survey of notable and popular wide-area coarse-grained task parallel executions and simulations.

The Great Internet Mersenne Prime Search (GIMPS) is the earliest known wide-area coarse-grained task parallel execution project delivered over the Internet [35]. This project attempts to discover large Mersenne prime numbers ($2^n - 1$). Since these numbers

are very large and grow exponentially, these tasks are computational intense but can be run independently from other users. Distributed.net is similar to GIMPS, but with different goals [36]. Distributed.net is known for distributing RSA Securities key cracking challenges in which the entire key space is searched using brute force methods. The key space can be trivially partitioned with non-existent data dependencies. SETI@home (Search for Extra-Terrestrial Intelligence) is an extremely popular task parallel execution that has been widely distributed, and is widely regarded as the first wide-area computational effort to gain traction in the general public [37]. This project partitions radio telescope data into frequency-independent portions that can be leased to individual volunteers and run with high concurrency as there is no need for synchronization among each partition.

Folding@home is a popular task parallel simulator dedicated to modeling and simulation of protein folding and discovering issues when proteins do not fold correctly leading to diseases [38]. The humanitarian effort of the project has garnered widespread appeal while being as easily accessible as task parallel executions. This effort has become so popular that ports of the software have been made to run on stream processors including graphics cards and home entertainment consoles such as the Sony PlayStation®3 [39]. The World Community Grid is a meta project performing simulations on a variety of humanitarian-focused programs such as human proteome folding and identifying potential drug candidates with the FightAIDS@home project [40]. ClimatePrediction.net distributes different climate simulation models to clients to predict future climate patterns and changes [41].

EP codes, with the distinguishing characteristic of no interprocess communication and messaging are performed on a range of different distributed computing infrastructures. In contrast, PDES programs are most often exclusively performed on computational resources that are tightly coupled for maximum performance. The following section details various execution platforms used for both PDES and EP codes.

1.1.3 Execution Platforms

As mentioned previously, the usual goals for parallelizing a simulation involve increasing model fidelity, integration of multiple simulators, and/or execution runtime reduction. In order to achieve these goals, PDES simulations are typically run on high performance computing (HPC) systems while recent wide-area coarse-grained task parallel simulations utilize resources afforded by the distributed nature of the Internet. Both of these distinct, yet related distributed computing paradigms are described next.

1.1.3.1 Tightly Coupled Resources

Cluster computing is the cornerstone of traditional high performance parallel and distributed computing infrastructures. Typical low-cost cluster computing systems are commercial off the shelf (COTS) machines with multiple processors or single package multi-core processors. Networking these systems together using low cost gigabit Ethernet provides good bandwidth and relatively low latencies for parallel and distributed simulation. These COTS clusters provide a balance between total cost and performance and are widely deployed.

Tightly coupled cluster computing systems trade low cost for higher performance than their COTS counterparts. These systems include faster and perhaps specialized

processors and large bandwidth, ultra-low latency switching fabrics from Infiniband, Quadrics, and Myrinet for example. Fine-grained PDES simulations can benefit greatly from these types of HPC systems that can reduce the latency in synchronization such as conservative barrier mechanisms and optimistic GVT calculations in addition to enabling faster messaging rates.

The very high end of HPC platforms involves supercomputing infrastructures. Some supercomputing facilities are simply constructed by scaling tightly coupled cluster computing systems, while others are custom designed from the ground up to include specialized processors, interconnects, and software including middleware tools and operating systems such as the IBM BlueGene system [42]. It is no surprise that these systems offer the highest potential performance and scalability for PDES codes [43]. Novel techniques have been applied for codes that are applicable to stream processors such as those found on general purpose graphics processing units (GPGPU) that are able to provide supercomputing-like performance with only a handful of graphics cards [44-46].

1.1.3.2 Loosely Coupled Resources

Grid computing through web services involves linking together various resources from different organizations and institutions to form a metacomputing platform [47]. These resources can be clusters, supercomputers, and even desktop computers [48]. The Globus Toolkit provides a standard set of services to create these kinds of systems [49]. There has been work in federating distributed simulations utilizing the High Level Architecture (HLA) over grids with IDSim [50] and SOHR [51-53]. Other related works include web-based simulation [54, 55], the Extensible Modeling and Simulation

Framework (XMSF) for web services [56], object request broker (ORB) based frameworks [57, 58], and a framework for Time Warp on grids [59].

Both PDES and task parallel codes are well-suited to run on traditional tightly coupled HPC infrastructures. There may not be as much need for coarse-grained task parallel executions and simulations for high bandwidth and low latency interconnects, however, fast processors and large memory pools are not a detriment to task parallelism.

A practical limitation of HPC infrastructures concerns their availability. Although grid systems alleviate some of the availability problem, access and restricted execution still exist. Allocated time on these systems is limited and is generally as not widely accessible as the loosely coupled metacomputing infrastructures described next.

1.1.3.3 Loosely Coupled Resources and Metacomputing

Recent advances in Internet-scale distributed computing have transformed the scope of certain computational work loads that, in the past, were reserved for large super-computing facilities and clusters of high-powered, dedicated machines. These wide-area distributed computing infrastructures are commonly referred to as public resource or volunteer computing platforms. Machines from all over the world form a virtual single super-resource offering computational capacity at the discretion of the user. Figure 5 illustrates a typical program lifecycle run on a volunteer computing framework.

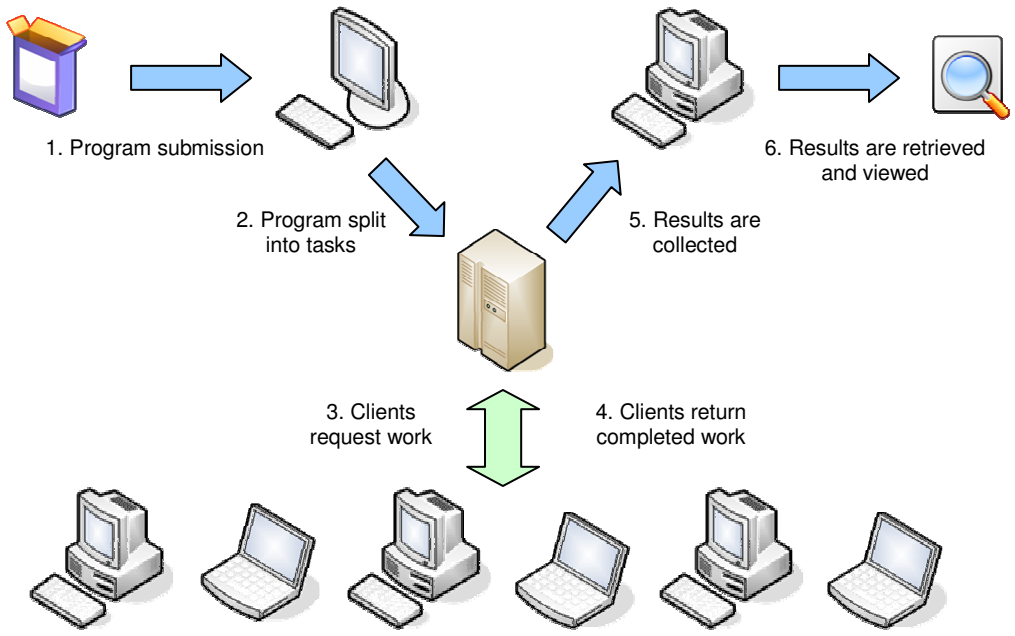

Figure 5: Volunteer Computing Task Life Cycle

Users may direct computations to only occur when their computers are idle (e.g., when the screensaver is active) or perhaps users may direct the client to perform computation at all times using one or more of the processing cores available on their system. Many of these projects have been discussed as task parallel executions or simulators such as distributed.net, SETI@home, Folding@home, World Community Grid, SZTAKI [60], and ClimatePrediction.net. Many of these task parallel volunteer computing projects are enabled by the Berkeley Open Infrastructure for Network Computing (BOINC) middleware software [61]. Other software solutions for volunteer computing include XtremeWeb [62], Unicorn [63], InteGrade [64], Harmony [65], DIRAC [66], and Xgrid [67]. These volunteer computing systems have offered very high computational throughput, rivaling even the fastest supercomputers [68]. BOINC-enabled projects have a combined throughput typically exceeding 1 petaflop [69] while

the Folding@home is the fastest distributed computing resource in the world exceeding 4 petaflops and over 350,000 active CPUs [70] mainly due to the large pool of stream processors including GPUs and the Sony PlayStation®3.

The emergence of grid and web services has allowed organizations and businesses to pool computational resources together to service workloads using existing infrastructure and machines that may not have originally been designated for providing processor cycles for these workloads such as desktop machines and laptops. Desktop grid computing is similar to the public resource computing paradigm with processor scavenging and utilizing idle-cycles, however, the scale, level of trust and implied security are different [71, 72]. These desktop grids provide cost savings for organizations by enhancing computational capacity for additional workloads or accommodating larger workloads not possible with the current employed infrastructure. These systems are typically on an institutional or organizational scale with implicit trust. In contrast to volunteer computing systems with wide-area applicability in both system compatibility and deployment, organizational desktop grids provide relatively higher speed interconnects and accompanying bandwidth for computational tasks. These desktop grids follow the same tradition of processor scavenging like volunteer computing projects but provide additional benefits such as reduced or eliminated need to replicate work for validation to counter Byzantine failures such as intentionally corrupt and falsified returns of results. With volunteer computing, results must always be verified and checked for accuracy to protect against be misbehaving clients, either unintentionally through hardware faults or intentionally through result falsification for expedited credit. In desktop grid infrastructures, varying levels of assumptions can be made about the quality

of results that are returned as there may be a higher level of trust and maintenance of hardware and the users of the systems. This type of computing potential has been widely researched, the most notable middleware tools and services are Parallel Virtual Machine (PVM) [73] and Condor [74]. Other frameworks include Entropia [75], and a master/worker variant of Condor named Condor-MW [76]. Task parallel simulation replication work on a network of non-dedicated resources with varying workloads is described in [77].

Both wide-area volunteer computing and desktop grids have certain disadvantages such as limits on types of applications that can be deployed. These infrastructures are almost exclusively tailored for task parallel simulations and executions and due to the public nature of most of these projects, simulations with sensitive data cannot be deployed. However, for the particular workloads for which these infrastructures are intended, they perform well if properly managed [78].

1.1.4 The Master/Worker Paradigm

Many of the loosely coupled distributed computing infrastructures borrow concepts from the master/worker paradigm. In this style of work distribution, the master oversees execution by assigning tasks to the worker pool. In public resource computing, atomic sets of work distributed to workers or clients are referred to as work units that contain partitions of simulation or perhaps an entire simulation replication. The master/worker paradigm imposes a restriction on communication, where no worker-to-worker communication may take place. The only valid communication links are between the master and the worker. In many volunteer computing infrastructures, communication is unidirectional as well, where service requests are only initiated by the client or worker

(e.g., “pull” mechanisms). This reduces complications and issues with firewalls as well as clearly delineating the master as a service and the workers as clients. Some master/worker task parallel simulations and executions take advantage of result compaction and simplification where the size of the input does not necessarily correspond to the output and amount of data that must be transmitted back to the master.

There are several inherent advantages a master/worker system offers due to the paradigm itself [76]. First, there is no burden on the user to schedule work or determine how to match work to clients. This is done by the master service; thus reducing complexity of the software that must be written. Second, client volatility is relatively easily addressed, as workers can be removed and added to the pool with much less hassle than a structured distributed infrastructure. Related to this point, fault-tolerance is more easily adapted for as programs for master/worker do not depend on the number of workers in the pool and the worker pool is expected to fluctuate over the course of execution. Moreover, if a worker fails to complete the assigned computation, it can simply be re-assigned to a different worker in the pool. Although a master/worker paradigm is subject to centralized points of failure, since the state of the entire workload exist on the master, simple checkpointing mechanisms can be implemented for failure recovery. Finally, load balancing is done on the worker end. In volunteer computing systems, if a worker becomes busy due to the user workload for other tasks, the worker can simply disengage from the distributed computation without fear of stalling or stopping the entire program.

1.2 Problem Statement and Research Challenges

Wide-area task parallel simulations and executions are predominately performed on loosely coupled computing infrastructures managed by software systems such as grid and metacomputing middleware. The advantages of these platforms are partly derived from the EP-style of computation and also follow from the advantages of the master/worker paradigm. PDES codes, however, have mainly been relegated to execute on traditional tightly coupled distributed computing infrastructures. Although these HPC systems offer the highest performance, there can be issues with deployment and readily available access.

The allure of harvesting computing cycles afforded by these metacomputing systems and desktop grids in particular for PDES computations is intriguing. The master/worker paradigm as described in the previous section offers capabilities such as reducing the burden on the users to run simulations, user-directed load balancing, system-level fault tolerance, heterogeneous machine support, the ability to share computing resources, and dynamic resource allocation and de-allocation, e.g., to add or remove processors during an execution. Paramount to all these advantages is the ability to utilize idle processor cycles that would otherwise be wasted.

PDES computations, however, are constrained further than typical task parallel applications due to LP and state management, time synchronization and message passing. By utilizing a master/worker driven metacomputing paradigm and addressing the special requirements of PDES codes, additional computational throughput capacity can be attained while retaining all of the associated benefits of a master/worker approach. The cost of this flexibility is overall performance. It is clear that PDES performance under a

master/worker metacomputing environment will never match that of a customized execution on a tightly-coupled HPC system. Such a performance gap exists between master/worker and conventional PDES systems on HPC platforms due to overheads inherent to the master/worker-style of work distribution. Under a master/worker PDES system, the state of each work unit must be stored on a master service for consistency and fault tolerance purposes. If a client with a work unit containing a portion of the simulation fails, the entire simulation will fail if the master service has no record of the last valid states. Therefore, each work unit lease incurs additional overhead not found in traditional PDES systems for checking out and checking in state variables that are part of the leased work unit to the worker. Moreover, messages cannot be directly sent in a master/worker system. Messages generated must be buffered on a master service temporarily before the proper work unit can download them when necessary. This indirect delivery of messages introduces additional overhead and message latency and degrading overall performance of the simulation. Finally, since the worker pool is heterogeneous all data exchanged must be serialized before transmission over the network. Additionally, this data must be de-serialized on the worker before it can be used. This data packing and unpacking introduces additional overhead typically not found in traditional PDES systems.

Although a performance reduction from conventional PDES systems is expected, a software infrastructure that provides additional computing throughput allows for other advantages such as more resilient and robust executions. Challenges for creating such an infrastructure are described next.

1.2.1 Portability

Most conventional HPC systems feature complete vertical homogeneity from the hardware to the operating system. Thus software on traditional distributed computing platforms seldom have to deal with portability issues. However, in public resource and desktop grid computing infrastructures, a variety of hardware architectures and operating systems must be accounted for in order to take full advantage of the available processing power in the worker pool. For task parallel simulations, this involves compiling the application for the target platform and encoding results in a platform-independent text or binary format. For PDES, the issue becomes more complex as the program code must be compiled for each possible target platform along with simulation state and messages that must be packed in agreed protocols or in a platform-independent fashion.

1.2.2 Node Volatility

Under volunteer and desktop grid computing platforms, it is expected that clients may drop from the worker pool. Measures are directly incorporated to deal with clients that cannot return results on time, fail to receive or send data, or provide incorrect results. However, the failure of one leased work unit is not detrimental to the overall progress of the entire project for typical volunteer projects. On the other hand, a failed partition under PDES can result in complete failure of the application. In traditional PDES, it is uncommon for a simulation to cope with node failure outside of a system that performs periodic checkpointing with a restart mechanism. PDES, under a volatile master/worker system, must consider volatility when partitions upon which other partiions depend are leased to a client that may fail. Controls must be implemented to ensure forward progress

in the presence of failed clients that never return a result or are too slow or may incur errors during updates.

1.2.3 Fault Tolerance

Related to node volatility is the issue of fault tolerance. As described earlier, under master/worker systems, fault tolerance is much easier to accomplish as failed workers are not systematically detrimental to the ongoing execution. However, a failure on the master service end can result in a complete execution shutdown due to a single point of failure. Additional fault tolerance protocols on the master portion must be considered when addressing PDES on these unreliable metacomputing frameworks as not only simulation metadata control information is stored under finite resources, but also simulation state information and messages.

1.2.4 Centralized Bottlenecks

In a typical modern PDES system, there are no centralized bottlenecks as time synchronization can be done asynchronously and in a decentralized fashion with regard to both conservative and optimistic mechanisms. Under a master/worker paradigm, by nature, there exists a master that exhibits some form of a centralized bottleneck. For PDES this is especially problematic, as the amount of data that must be moved is much larger than those found in traditional task parallel applications.

1.2.5 Bandwidth and Latency Concerns

With a large amount of data that is transmitted over the course of a PDES, bandwidth and latency becomes a concern when operating under a desktop grid infrastructure. New protocols and policies must be devised for both conservative and

optimistic synchronization to reduce congestion, preserve useful computation and avail work unit locality on the workers to the distributed simulation.

1.2.6 Load Balancing

Load balancing in traditional PDES is a difficult problem and is not commonly found in most monolithic PDES codes and even many run-time infrastructures. Systems that do have load balancing capabilities often use pre-computation or sequential runtime data first to determine event and computational density in order to partition and allocate resources effectively during a distributed run [79]. Dynamic load balancing schemes are most often specialized for their application domain and may not be portable across all PDES codes [80], while generic dynamic load balancing schemes do not support dynamic resources and heterogeneity among nodes [81, 82]. In contrast to traditional PDES, load balancing in PDES under a master/worker system must be generic enough to support all PDES codes applicable to this paradigm as well as support dynamic resources along with non-homogeneous workers. Load balancing with respect to master/worker systems can be split into two parts. First, the clients themselves are load balanced through a combination of the master matching available work to idle clients including matching software requirements with the available hardware along with clients themselves disengaging from the execution if they are no longer available for computation. Secondly, load balancing can be applied on the master side as well, where state and message storage for LPs can be migrated between high and low load servers. Additionally, new resources whether they are master services or workers can be immediately integrated into the running simulation.

1.3 Research Contributions

The research contributions addressing the issues and challenges facing integration of non-traditional execution platforms, the master/worker paradigm, and PDES are as follows:

- **Master/worker architecture for PDES**. I have developed an architecture to address the challenges and issues facing implementation and execution of PDES codes across loosely coupled distributing computing infrastructures such as volunteer computing and desktop grid systems. This architecture delivers fully reproducible results in a metacomputing environment through the development of portable, scalable, load-balanced, fault-tolerant, idle-cycle capturing services and protocols. I have proposed a set of fault-tolerance protocols for the master services to provide robust execution in the presence of failures. Additionally, I have developed extensions to the master/worker framework to allow an integrated and insulated execution environment for simultaneous PDES and task parallel simulations. The mechanisms developed allow any number of PDES simulations and replications to be run concurrently with task parallel simulations.
- **Analysis of Portability Approaches and Impact on Performance.** I have analyzed different approaches to portability from web services to highly portable libraries, showing their strengths and weaknesses with regard to architecture independence and performance as it applies to a master/worker PDES architecture. I have addressed scalability concerns under a master/worker PDES architecture by developing protocols to distribute the set of services under the master portion of the paradigm allowing dynamic allocation of storage resources as needed. An empirical study comparing a monolithic universally portable system to a slightly

less portable distributed architecture was performed and presented with quantitative differences between the two approaches. I have shown significant speedup by reducing excessive artificial overhead from a widely accessible web service approach and providing simulation capability for large-scale PDES through the use of distributed master services without large sacrifices in portability.

- **Performance evaluation of a master/worker PDES system.** Utilizing both synthetic workloads and real world applications, I have performed various empirical studies on master/worker PDES systems. I have characterized and evaluated key PDES properties such as lookahead, granularity, and computation to communication ratio for a master/worker environment. Understanding these characteristics can better classify which PDES applications are best suited for a master/worker execution. Moreover, for master/worker systems I have developed underscore the need for a more relevant metric by comparing and contrasting the amount of processor time spent in actual PDES computation versus overhead times associated with the master/worker environment. The proposed metrics provide a breakdown of each major component, and a profile indicating what portion of the total processor time dedicated to executing simulation application code as opposed to overhead. These metrics provide more useful and relevant information than traditional speedup metrics typically found in PDES performance studies, and are applied to different performance tests under a master/worker PDES system. The results from these empirical studies show and validate the impact of key PDES properties on overall performance. The viability

of a PDES system under a shared loosely coupled computing resource is demonstrated. Most importantly, the performance studies show PDES codes that are the most conducive to a master/worker execution leading to a clear classification of expected performance (e.g., excessive overhead or acceptable overhead) according to inherent model characteristics.

- **Conservative execution optimizations**. In order to reduce the amount of intrinsic overheads involved in a master/worker PDES computation, I have proposed several optimizations that have been applied and evaluated to the master/worker PDES architecture. First, the design of a caching mechanism for storing recent simulation states on the workers along with various eviction policies is discussed and incorporated. Second, scheduling policies for work units are described where lookahead and other time information along with runtime statistics are exploited to better prioritize partitions of work to clients. Third, a mechanism for overlapping communication with computation is proposed to efficiently pipeline simulation state updates. Similarly, a variety of techniques for masking communication costs associated with messages is designed and evaluated. Finally, a protocol for pro-active message updating is described. Together, I have shown that these optimizations significantly reduce the performance gap between master/worker and traditional PDES systems using synthetic workloads and a real-world application.
- **Optimistic execution mechanisms**. Optimism on a master/worker paradigm across volatile computing platforms presents new challenges. Due to the centralized nature of metadata, state and messages, traditional Time Warp

concepts must be adapted to fit this paradigm. I have developed new techniques to allow optimistic executions to be performed under this metacomputing paradigm. Master/worker PDES operates on the principle of leasing execution windows for workers to process, so methods must be developed on determining the proper length of these windows even under pure stochastic simulations with no lookahead defined a priori. I have proposed two new rollback mechanisms to effectively deal with window-based leases and messages delivered via proxy. Issues such as delayed rollbacks due to no peer-to-peer connectivity, unique message identification across distributed master services and causality linkages are analyzed. These protocols handle rollbacks on the master services, as well as preserving the maximum amount of work already completed. Additionally, adaptive tuning of time window lengths and adaptive state saving mechanisms are proposed and evaluated.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents a portable approach to master/worker PDES through web services. A scalable and concurrent approach to master/worker PDES and task parallel simulation with specific focus on desktop grid architectures is discussed in chapter 3. This is followed by the design and evaluation of optimization techniques to reduce intrinsic overheads in conservatively synchronized master/worker PDES in chapter 4. Chapter 5 details new approaches to Time Warp given a master/worker infrastructure for parallel simulations.

Chapter 6 provides a summary of the work presented in this thesis along with possible future directions of master/worker PDES systems.

CHAPTER 2

A WEB SERVICES APPROACH TO MASTER/WORKER PARALLEL DISCRETE EVENT SIMULATION

In a wide-area metacomputing distributed computing infrastructure such as public resource computing, the pool of available resources in the form of workers is not guaranteed to be homogeneous. It is expected that there will be a variety of workers used to form the resource pool including machines ranging from laptops to high performance computing platforms. Each of these clients may have a varying system architecture and operating system complicating application deployment and data transfers. Providing for cross-platform compatibility is a challenge with the variety of resources that can partake in a distributed computing project. One approach is to employ nearly complete architecture and language independence through the use of web services.

Web services are a set of principles encompassing system design, component interaction and communication protocols for interoperable exchange of information over a network. These services are called web services because the network used for this type of software is commonly the Internet along with the use of pre-existing protocols driving the World Wide Web. Web services can offer functionality similar to traditional Remote Procedure Call (RPC) client-server interaction [83], but are often used to create larger systems under a Service-Oriented Architecture (SOA).

Through the use of open standards and open source implementations, web services provide accessible interoperability and a set of standards for building client-

server applications. The SOAP protocol is typically used as the underlying messaging framework to send XML-encoded messages between the server and clients. SOAP is the preferred method for encapsulating data for transmission across the network under web services due to its application and language independence, allowing for ease of implementation and incorporation into applications utilizing web services. The Web Services Description Language (WSDL) describes the services provided by the server for the clients. SOAP messages are often sent using the Hypertext Transfer Protocol (HTTP). The advantage of using HTTP is that service requests can be provided through security measures such as firewalls, since HTTP access is usually unrestricted.

2.1 The Case for Master/Worker PDES in a Universally Accessible Web Services Framework

The ability to provide a simulation framework that is openly accessible regardless of the programming language, operating system, or underlying machine architecture allows a simulation application developer to create applications that can be essentially run anywhere with network access [84].

With large-scale simulations requiring extreme amounts of computing power via supercomputers or grids, preparing a simulation to run across a massive number of processors is both time consuming and potentially expensive if computation time is leased. In most conventional PDES executions, if a node fails, the entire simulation will fail with no means to salvage the execution unless a checkpoint/recovery system is implemented. A master/worker system allows a level of robustness for parallel and distributed simulations for dealing with node failure as well as distributing simulation

load across the available client pool without having to explicit implement a checkpointing system, as node dynamics is implicitly understood under a master/worker execution.

In addition to the ability to cope with node failures, a major feature of these non-traditional distributed computing infrastructures spread across shared machines is the ability to run simulations in the "background." Machines running as workers can also be processing other jobs. This enables simulations to run on unreserved and potentially volatile machines, while contributing to the overall distributed simulation computation. With the accelerated acceptance of multi-core desktop and laptop machines in recent years, running computations on a free core may not significantly impede on a user's interactivity with their foreground processes while allowing these background computations to run at nearly full speed in a transparent fashion.

Although a master/worker system permits flexibility with regard to simulation execution on different platforms, load balancing and fault tolerance, the framework is not suitable for every type of simulation. This style of execution is best suited for applications where a significant amount of computation can be handed to a client for execution. This is most likely to occur in large-scale simulations with a high degree of parallelism. As is true for traditional PDES executions, tightly coupled simulations with much global communication may not be well suited for a master/worker distribution of work. Similarly, distributed simulations with a low amount of parallelism or where only a small amount of computation can be done before the LPs passed to a client must block are also better suited for conventional or perhaps sequential execution. An important contribution of this research is to quantify these concepts to determine the range of applications where the master/worker paradigm is well suited.

2.2 A Master/Worker Architecture for PDES based on Web Services

The emergence of web services has allowed applications to exploit open standards-based interoperable communications over the Internet. A master/worker metacomputing framework, Aurora, extends these principles to PDES through the use of web services [85]. While conventional web services have emphasized interoperability over performance, the Aurora system was built with high performance as a priority while providing uncompromised levels of interoperability on the language and machine architecture levels. The Aurora system itself was constructed with extensibility in mind as different pieces of the system are modular and can be replaced without a significant amount of change to the codebase and API.

2.2.1 Conceptual Overview

The Aurora system applies a PDES execution to the master/worker paradigm while leveraging the advantages of web services. Following standard accepted practices, the parallel simulation program is assumed to consist of a collection of LPs that communicate exclusively by exchanging time stamped messages. LPs, with associated data structures, as discussed below, are clustered into *work units.* A work unit is the atomic unit transmitted between the server and clients. In a master/worker paradigm, the master controls the global available work pool and manages the overhead associated with each work unit. Worker threads or processes perform the necessary computation on these work units and return results to the master. This cycle continues until all of the work units are exhausted or some other terminating condition is met. For example, suppose a simulation consists of two work units that can communicate with each other via links

with symmetric lookaheads of 0.5 time units. Figure 6 illustrates two clients interacting with the master service under this simulation scenario.

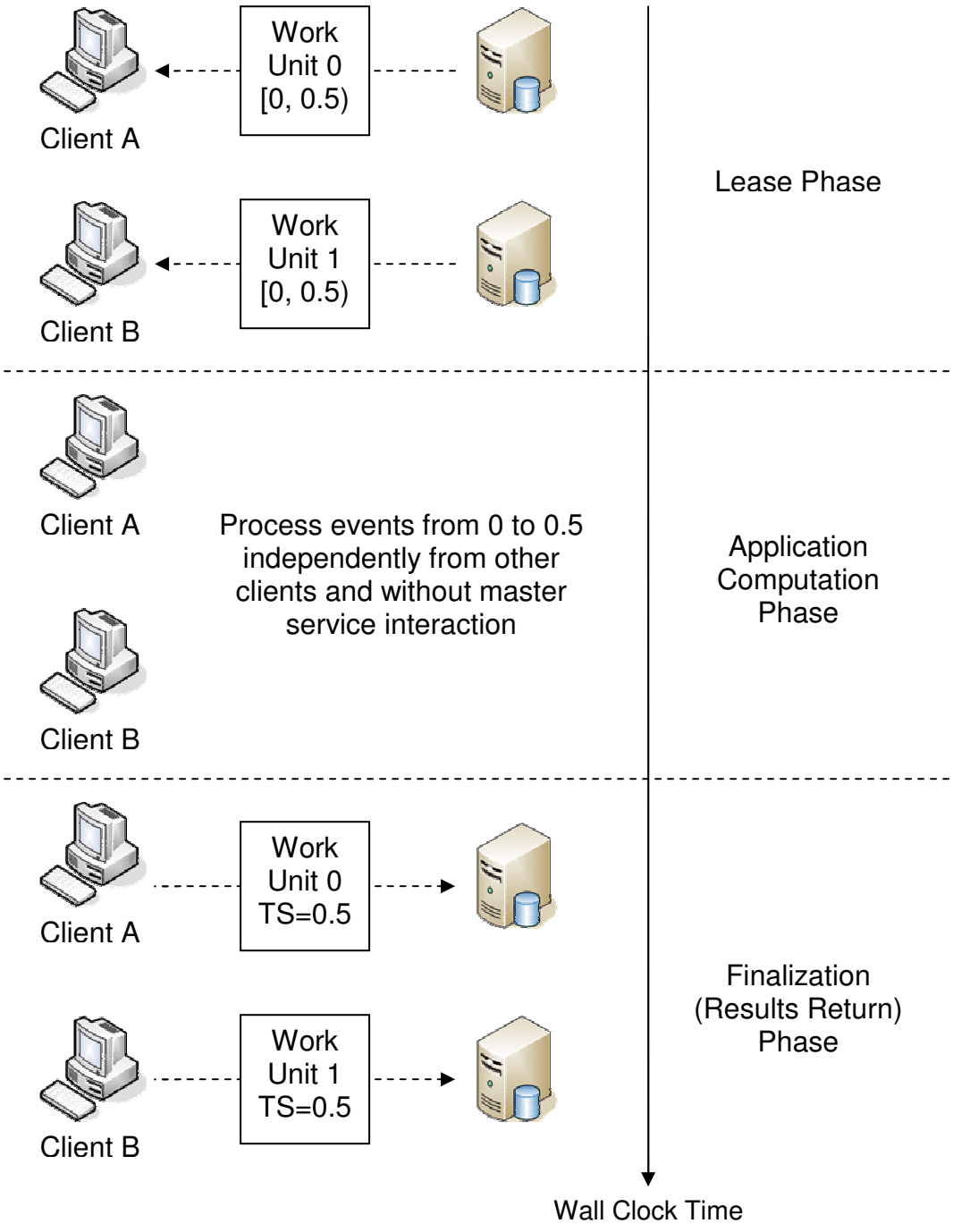

Figure 6. General Master/Worker PDES Interaction Overview

Each client (or worker) contacts the maser service requesting work, valid work units are leased to clients during the lease phase. For this example, work unit 0 and 1 with specific lengths of simulation time or time windows are provided for the clients. Under conservative synchronization, these time window lengths would be guaranteed to not induce causality errors. Simulation data associated with this time window such as state vectors and input messages are downloaded by the clients. In the next phase, each client runs independently of every other client in the system including the master service where the application code of the simulation is executed for the time interval given. Once the application computation completes, then the final work unit states and any messages generated are sent back to the master service during the finalization phase.

In traditional conservative PDES, the simulator must ensure that events are processed in strict time stamp order to avoid violating the LCC. Consequently, synchronization algorithms are used to calculate guarantees such as LBTS to regulate which events are safe to process. In optimistic PDES, the simulator may allow violations of the LCC to occur but must recover from such errors. The design of the web services version of Aurora is built around a centralized conservative synchronization approach for the ease of initial development.

In addition to providing time management services, the Aurora system bridges the concept of work units and LPs. LPs generate messages for other LPs that may be local to it (i.e., residing within the same work unit) or destined for remote LPs residing in other work units. This is a major departure from other systems in the area of loosely coupled distributed computing infrastructures. In massively distributed computing projects such as SETI@home, portions of work sent to client machines do not require communication

between leased units of work due to the EP style of computation. The Aurora system keeps track of messages that are generated from source work units and correctly distributes them to destination work units.

The Aurora system also includes an authentication and metadata module to keep track of work units. Work units are designated as *available* or *leased*. An *available* work unit means that there are no clients currently performing computation on that work unit and is ready to be released to the next client request. Work units are marked as *leased* when the server releases it to a client. If a client is issued a work unit, a global unique key is assigned along with the work unit. This allows work units to be issued more than once for fault tolerance purposes, or in the case of abundant heterogeneous client machines, to issue the same work unit to multiple machines with the hope of receiving results more quickly if the relative execution speed among the different machines cannot be predicted, e.g., due to contention from other users. The authentication and metadata module is also crucial for allowing optimistic synchronization.

2.2.2 Communication Framework

The default communication mechanism used in Aurora is based on SOAP, providing maximal support for interoperability among heterogeneous computing platforms. The Aurora system, however, is designed to support different communication mechanisms. When used on a tightly coupled parallel computer, communications based on MPI may be used. Alternatively, sockets can be used in homogeneous networked environments. The communications interface is a thin layer that is invoked when the client and server agree on a specialized transfer protocol other than the default SOAP transport during the handshaking phase of client initialization. Specialized

communications may improve simulation performance in certain situations where SOAP-encoding and XML-parsing can be bypassed when an alternative messaging mechanism is available. The Aurora system based on web services uses SOAP exclusively for communication.

One of the perceived disadvantages to developing a PDES framework under web services is low performance [86]. Due to the inherent nature of transmitting XML-encoded data, an optimized PDES engine should outperform any simulation framework based on web services. The gSOAP toolkit [87] is a mature, active, open source web services toolkit designed with performance as well as language and machine architecture interoperability as priorities. gSOAP is intended for applications in C/C++ but can be bridged to languages such as Fortran and Java with JNI. gSOAP supports many industry-standard web services protocols including SOAP 1.1/1.2, WSDL 1.1, and UDDI v2.

The gSOAP toolkit has shown low-latency and high performance by utilizing various techniques including streaming XML parsing, Base64/DIME encoding, HTTP chunking, HTTP compression, and HTTP Keep-Alive. gSOAP coupled with these techniques has been shown to outperform Java RMI for binary-encoded matrices with latencies under seven milliseconds and as low as one millisecond [88]. gSOAP also exhibits good end-to-end performance in sending arrays of different primitives and low serialization and deserialization times [89]. Since the Aurora system is intended to be application-independent, it is oblivious to the actual data contained in state vectors and messages sent between LPs. Consequently, the performance of binary transmission is critical. gSOAP exhibits relatively low overhead for encoding and decoding binary data in small messages.

2.2.3 Master Service Design and Implementation

The control and management algorithms of the system that work together to form the master are components of the Aurora server. This service contains the actual web services routines and three major modules for time management, work unit and logical process bookkeeping, and client authentication and metadata information management.

Figure 7 shows the interaction between the web services routines and the various modules within the Aurora server. Some calls are exposed to the application developer to initialize the Aurora system with values for a particular simulation. Configuration parameters include the maximum number of logical processes that can be leased as a work unit and initial logical process state and lookahead values. The server-side application is lightweight as no actual simulation computation is performed on the server.

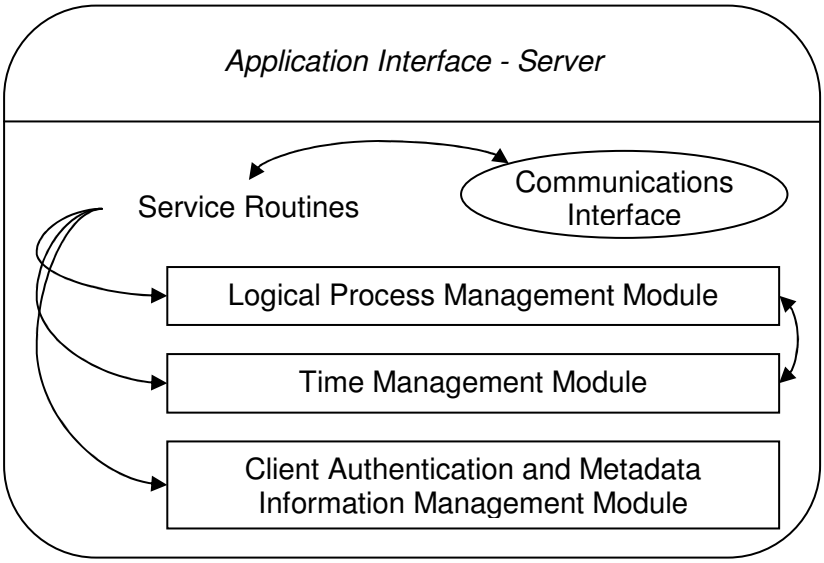

Figure 7: Web Services Master Service: Aurora Server

The Aurora server architecture includes a module to support fault tolerant execution. This module allows the re-release of LPs if certain criteria defined within the

module are met; mechanisms to manage duplicate results are also included. Also, other modules can be replaced or extended. For example, the client authentication and metadata information manager can be expanded with the WS-Security module recently added to the gSOAP toolkit.

2.2.3.1 Logical Process Management

The logical process manager keeps track of application defined LPs. The simulation application may aggregate many LPs into a single Aurora work unit. An Aurora work unit is instantiated by the server-side application containing the initial state of the LPs. Each work unit stored in the Aurora server includes one or more state vectors containing simulation variables, an event list, and an input and output message buffer associated with that work unit. Due to the application-independent nature of a metacomputing master/worker system, the LP state vector is stored as a contiguous block of memory that is packed and unpacked by externally defined procedures. The message buffer for each work unit consists of a table of messages destined for the LP that have been received from other LPs. The event list contains locally scheduled messages. Each message is wrapped in a data structure providing information such as the message timestamp, destination, and size of the packed message.

A work unit includes an LP or collection of LPs with associated buffers and metadata such as the LBTS value that is transmitted to a client as an atomic unit. When a work unit is executed and successfully returned to the server by the client, the LP manager first updates the state vector of the LPs contained in the work unit stored in the server. After this completes, the server performs a process known as *binning*. The messages packed in the output buffer from the returned work unit are scanned and placed

into the correct input buffers of the destination work units. During this process, the LP manager re-organizes the input buffer of the returned work unit, freeing memory for any message buffers that can be released.

Upon the successful update of a work unit with a new state vector and shuffling of messages to the proper buffers, the Aurora LP manager calculates the minimum timestamp over all input messages for the updated work unit. The server then performs an LBTS time management computation before the success status code is returned to the client. The forced LBTS update after each completed work unit guarantees that the server can lease the maximum amount of work to future leased work units.

2.2.3.2 Conservative Time Management

The web services based Aurora time management system provides support for conservative synchronization. Because time management computations are performed at the server, a simple, centralized algorithm for computing an LBTS-like value of future messages that may be received by an LP is sufficient; distributed algorithms would be required for servers utilizing multiple processors. In Figure 7, the interaction between the time management module and the logical process module denotes the synchronization calculations each time a work unit is returned to the server after a client completes its execution.

Under conventional PDES systems, time management can occur using any number of algorithms such as distributed reductions, null messages, deadlock detection and recovery mechanisms, etc. Master/worker systems offer a distinct advantage due to the centralized nature of the master services. The master service will always have the most up-to-date simulation time information along with timestamps of any messages that

have been generated. This allows a direct centralized time management approach under conservative synchronization.

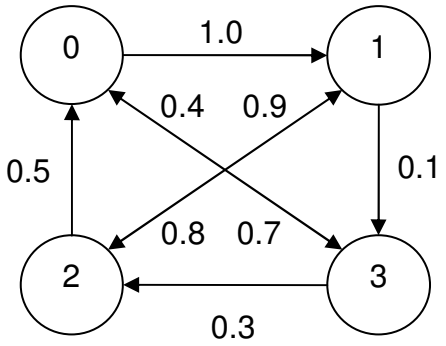

Figure 8: Example 4 WU Connectivity Graph

Table 1: Corresponding Connectivity Matrix

	0	**1**	**2**	**3**
0		1.0	-1	0.7
1	-1		0.8	0.1
2	0.5	0.9		-1
3	0.4	-1	0.3	

Through information provided a priori to the master services such as lookahead and connectivity between work units, a global view can be created for throttling the simulation and adhering to the LCC. Figure 8 shows an example connectivity graph illustrating message paths between four work units within the simulation. The master service is able to create a lookahead connectivity matrix where lookaheads are stored as shown in Table 1 with positive values representing lookahead and negative values denote no connection. This information is used to create Minimum Emittable Time Stamp (MinETS) tables during each work unit lease to provide simulation execution windows.

The MinETS computation is similar to LBTS computations. Traditional LBTS computations are simply the smallest possible timestamp that a logical process can receive in the future. In a master/worker PDES system with centralized time management control, incoming time information is required over all possible input channels to a work unit, however, instead of minimum time stamped messages, the stored simulation time or current execution window end time is used.

$$MinETS_i = \min_{\forall j} \left(S_j + LA_{ji} \right) \qquad (2.1)$$

Equation (2.1) defines MinETS at work unit *i*, where the minimum emittable time stamp of all input channels to work unit *i* are computed by adding the current simulation time (*S*) to the lookahead (*LA*) between work units *j* and *i*. This MinETS value at work unit *i* is used as the end time or safe processing bound for a lease execution time window.

Table 2: Example MinETS Snapshot

	0	**1**	**2**	**3**
Simulation Time	0.4	0.9	0.3	0.7
MinETS @ WU 0		-1	0.8	1.1
MinETS @ WU 1	1.4		1.2	-1
MinETS @ WU 2	-1	1.7		1.0
MinETS @ WU 3	1.1	1.0	-1	

Workers can execute up to the temporary end time provided with a guarantee that there will be no other incoming messages during this time window that would cause incorrect execution requiring a rollback recovery mechanism. For example, the master service would have a MinETS snapshot similar to data shown in Table 2. From this table, execution time windows can be created. Work Unit 0 is at simulation time 0.4 with

minimum emittable timestamps of 0.8 and 1.1 from work units 2 and 3, respectively. Taking the minimum of these entries gives a lower bound of 0.8 allowing the master to initiate a lease for work unit 0 from time 0.4 to 0.8 as no messages can be delivered to work unit 0 within this time. This centralized approach to time management simplifies conservative synchronization eliminating the need for mechanisms requiring messages and updates.

There is potential for increased performance through an optimistic synchronization algorithm. Due to the inherent nature of a master/worker system, LPs can be leased at will to any client that is available for execution. Furthermore, the results returned by any client do not have to be used. Depending upon how results are cached on the server to accompany an optimistic time management system, error recovery can simply restore state from a known correct state instead of using a rollback recovery system. The focus under this web services based master/worker system is on a conservative execution.

2.2.3.3 Client Authentication and Metadata Information Management

The Aurora server will assign a work unit to any client that meets the proper requirements set forth by the simulation application. A leased work unit must be uniquely keyed and marked appropriately as clients may be returning and requesting work units at any time during the lifetime of the server. Other metadata such as references to output buffers constructed for the leased time window for a particular work unit must be stored in case of a work unit re-release in order to properly manage memory upon the return of a completed work unit.

2.2.4 Worker Design and Implementation

The workers in the master/worker paradigm are implemented by Aurora clients. Each Aurora client pulls the necessary work unit information from the server through web service requests, executes the simulation implemented by the application developer, and uploads state vector and output message buffers back to the master Aurora service upon completing execution according to the specified time management scheme. A conservative execution will process all events within a work unit with timestamp less than the LBTS value computed for the work unit.

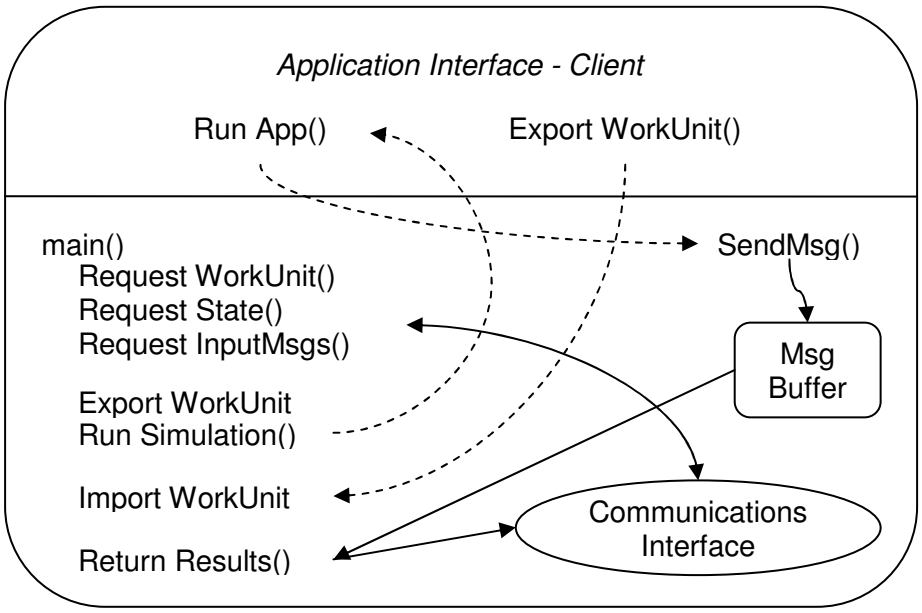

Figure 9: Aurora Client Design

Figure 9 shows the interaction and data flow between different function calls and certain key data structures within the Aurora client. After initialization, the Aurora client performs a series of web service requests to the server. During the handshake phase, the client requests an available work unit from the Aurora server as well as possible communication modes in addition to the default gSOAP transport. Once the client

receives work unit availability confirmation from the server, the state vector and all of the incoming messages for the leased execution time window are downloaded from the Aurora server. Once this information has been received from the Aurora server, the client can operate autonomously from the master.

The incoming messages for the work unit are queued automatically for retrieval by the client in timestamp order. The state vector is uploaded to the application (data moving between the Aurora client and the actual simulation application are shown by dashed arrows in Figure 9), and the Aurora client invokes the application simulation loop. During the simulation, the application may generate messages that are destined for LPs residing in other work units. The Aurora client provides a message send interface where all messages that exceed the LBTS time of the work unit are buffered for upload to the server when the simulation completes. After the simulation completes, control is returned back to the Aurora client and the final state vector is imported back into the Aurora client. The state vector must be correctly packed for proper Base64 encoding and transmission to the Aurora server. Once the state is finalized, the state vector and output messages are packaged and sent to the server.

2.2.5 Additional Requirements for Master/Worker PDES

Under a master/worker PDES system, the simulation modeler must specify a lookahead connectivity table as described previously. This is required for the system to determine work unit to work unit connectivity to create guarantees and simulation time windows. Additionally, the modeler must partition the problem into work units. Each work unit may consist of a single LP or multiple LPs aggregated together. Currently this is a manual process and specific care must be taken to ensure that each work unit is

"large enough" or computationally significant to ensure enough processor time is spent in application code compared to overheads. The impact of work unit granularity is quantitatively evaluated in performance evaluation sections. Finally, the modeler must implement serialization and deserialization routines to ensure that data structures and messages generated by the simulation application are portable across different machine architectures and operating systems. Under the web services framework, much of this is automatically done through XML encoding and WSDL definitions, however, care must be taken with custom messages generated to ensure cross-platform compatibility. These limitations and issues are explored further in chapter 3.

2.3 Fault Tolerance for a Web Services Based Master/Worker PDES System

The Aurora system provides a transparent fault tolerance system for simulations that exploits the inherent state and message saving that is necessary in a master/worker paradigm. By leveraging this mechanism, most of the support and modifications are only needed on the back-end services. The two major approaches used in the Aurora system for fault tolerance are replication and checkpointing.

2.3.1 Resilience to Client Failure

Due to the assumption that clients are not guaranteed to return a result, one of the requirements of creating a simulation package definition is to specify a *deadline*. These deadlines are essentially wallclock runtime limits for any leased work unit for that particular simulation. If the back-end services do not receive a work unit return within this time limit, it is assumed that the client has failed whether this is due to the client crashing or the computation exceeding the stated wallclock time limit. If a work unit

return is not received by the deadline, this work unit is re-issued to another client. If the client attempts to return the work unit after their deadline period, the results are simply ignored and discarded.

A total reliance on deadlines for client failure and recovery can lead to less than optimal performance in certain situations due to long wait times and possible lease failures due to the unavailability of work units. A simple solution to this problem is to utilize a heart-beat mechanism where the client periodically provides a progress report to the back-end at application-defined time intervals. This option may be detrimental to performance in situations where the amount of computation per work unit lease is low and the addition of more service requests can degrade back-end responsiveness.

2.3.2 Resilience to Server Failure

Server failure is a more difficult problem to address than client failure due to the back-end possessing the simulation state and possible transient nature of the system at the time of failure. The approach used in the web services based master/worker PDES system is described in the following sections.

2.3.2.1 Checkpointing and Restoration

The internal metadata and states must be saved in case of a server side crash. There are two logical times at which this can be done: the checkpoint after each work unit return or the checkpoint after a LBTS advance. Checkpointing after each work unit return can incur high overhead due to frequent checkpointing but has the advantage of being able to store the latest computational results. The latter has the advantage of lower

overhead at the cost of possibly sacrificing recent computational results. Checkpoint on LBTS advance was implemented in the web services based Aurora system.

In addition to work unit and message state checkpoints, metadata has to be stored as well. Information such as key tables and time information are serialized and written to a hard store such as a common file on disk. Recovery from a failure is handled by reading this information from the file, and re-populating metadata tables and saved state.

2.3.2.2 Replication

To augment the robustness of the Aurora system to handle runtime failures, replication is used where the Aurora server was mirrored to different machines. One server was designated as the primary server, while the other servers were secondary servers forming a cyclic ring based on a specified sequence. Clients are also given this information on the availability of servers. If the primary server is unresponsive, the clients will attempt to contact the next server in the ring. If a secondary server receives a work unit request, it broadcasts a vote to the other servers in the ring for promotion as the primary server. If during this phase the primary server did not actually crash and provided a delayed vote return, perhaps due to a network issue, the secondary server that issued the vote will assume the primary server has crashed. The newly designated primary server will then periodically send messages to the previous primary server indicating leadership change in cases where the primary server did not actually crash but is experiencing slowdown issues.

The new primary server will use the common checkpoint store to populate metadata and state upon promotion. Once the simulation state has been restored, execution can continue and work units can be leased. Since the simulation has a

possibility of rollback, work units that may be returning that are out-of-sync are simply discarded. In the web services based Aurora fault tolerance system, no transient server failure is assumed on the client side, thus once the client assumes the server has crashed, it will no longer try to contact it in the future.

2.4 Metrics of Performance

Traditional performance metrics such as speedup do not fully capture the benefits of utilizing a master/worker desktop grid based simulation system. It is understood that PDES codes on a master/worker infrastructure will likely never be as fast as tightly-coupled cluster systems which will always provide the maximum potential performance with minimal overheads.

For PDES master/worker desktop grids, we attempt to measure throughput by contrasting the total processor time divided into four major components:

1. Deferred time: no available work units (idle wait time)
2. Work unit setup time: the time required for lease acknowledgement, work unit state vector and message download and unpacking
3. Application runtime: actual PDES code execution time
4. Work unit finalization time: the time required for work unit state vector and message serialization and upload and consistency convergence

The first portion of processor time is the deferred wait time, or idle wait time of the client. It is possible for a client to request work when no work is available. Time spent spinning in the work unit request loop can be classified as overhead time. This is a non-deterministic value where a variety of factors can contribute to this such as

heterogeneity of computing resource (i.e., a pool of slower machines), network congestion, or service overloading as a result of improperly anticipating the load of the simulation. This value can be minimized, although never eliminated as work unit request-acknowledgement is necessary, if the proper partitioning of the model is performed, a correct choice for work unit scheduling (e.g., which work unit to schedule and where to schedule it) along with the proper resource allocation for back-end services.

The second portion of processor time is the work unit setup time, which is another component of overhead. After the master service approves a work unit lease to a client, the client must contact and download the proper state and messages from storage services on the back-end system. Once downloaded, the client must then unpack the simulation state vectors and messages that may consume a significant portion of processor time if numerous state vectors or messages exist. Scaling the back-end services appropriately for the size of the PDES code can reduce this time, but is also restricted by the bandwidth available between the client and storage services.

The application runtime component provides a quantitative metric of how much processor time is spent performing application code rather than various overheads within the master/worker system. The proportion of time spent in this phase compared to overheads provides a clear picture of the efficiency in running an application under a system such as Aurora.

The final component of characterizing processing time is the application wrap-up and finalization phase. This includes the time spent re-packaging all state vectors, messages, and sending all data back to the back-end services once the consistency convergence is approved by the back-end system. The current Aurora system attempts to

parallelize this phase by simultaneously packing state and messages. Much like the second component (work unit setup time), this overhead can be minimized but not eliminated through properly instantiating an adequate number of back-end services with sufficient accompanying bandwidth.

By dividing the processor time into four distinct components, we can more appropriately measure the amount of total time that is being utilized by the actual simulation computation compared to the time due to overhead either inherent to the design of the master/worker paradigm or implementation.

Although runtime reduction is desirable for PDES codes, benefits such as off loading less time-critical simulations to idle machines, run replication support, or providing application insulation in an integrated distributed simulation can offer benefits other than strict speedups. These metrics allow the measurement of pure application runtime with respect to the total runtime, providing a clearer picture of the overall work performed in the context of total processor time.

2.5 An Analytical Performance Model

As mentioned in the previous section, in a traditional PDES execution performance can be classified by metrics such as speedup relative to a sequential implementation. Because Aurora is a high-throughput computing system and may execute on non-dedicated hardware, speedup is not the most appropriate metric. The *efficiency* of the simulation given the master/worker infrastructure where client machines are able to contribute a certain amount of processor time to perform simulation computations can quantify how well a simulation is executed across the available

resource pool. The fraction of time used performing application computations as opposed to overhead computation or wasted through idle processor cycles is the basis behind the efficiency a PDES application can achieve across this type of infrastructure. Using the metrics as presented in the previous section, we can create an analytical model for performance for a monolithic metacomputing master/worker PDES system under web services.

The first major parameter that negatively affects efficiency of a simulation is overhead. Overhead in the Aurora system arises primarily from the time required to transfer work units between the client and server machines; the amount of simulation computation performed by a client once it receives a work unit also greatly impact efficiency. These factors, in turn, are affected by three principle parameters:

1. State vector size
2. Number of input messages (messages sent from server to client) and aggregate input message size
3. Number of output messages (messages sent from client to server) and aggregate output message size

These three characteristics directly affect the overhead of shipping the work unit from the server to the client and back. In the current implementation, the state, input messages, and output messages are encoded in Base64 then sent using a SOAP message. The larger each of these parameters becomes, the more overhead that is incurred for each work unit release and return. The number of input messages is included due to the processing time required on the server to construct the correct input buffer for the work unit being released to the client, excluding any messages that cannot be executed in the

current processing cycle because they are too far into the simulated future. Processing time is also required on the client side to process the packed block of memory containing input messages and queue the messages for the simulation application. The number of output messages is included as the client must construct a suitable packed block of memory for Base64 encoding. The server must also process the output buffer from the completed work unit and bin the messages to the appropriate destination LPs.

Overhead can be calculated as:

$$T_o = \frac{S_s + S_i + S_o}{T_{rate}} + \frac{N_i}{B_{rate}} \qquad (2.2)$$

where S_s is the average sum input and output state vector size, S_i is the average input buffer size, S_o is the average output buffer size, T_{rate} is estimated transfer rate for the selected communication mode, N_i is the average number of input messages, and B_{rate} is the estimated server message processing rate for binning output messages into the correct input buffer. This equation measures the time to transmit messages and message processing time overhead.

We can now construct a model for approximating that time a simulation application runs under a conservatively synchronized Aurora system. Lookahead is a simulation characteristic that can affect the efficiency of a simulation tremendously. If the lookahead is too small, client concurrency will be reduced whereby instead of doing useful work, there will be increased shuffling of state and message buffers between the client and the server. In a centralized conservative time management system, the lookahead plus either current simulation time or the minimum of input message times determines the LBTS value. For each work unit released to a client, the client may only execute from the current simulation time up to the LBTS value.

The partitioning of the simulation's LPs into work units also plays a large role in the efficiency of an Aurora simulation. If the number of available work units is smaller than the number of available clients, clients will become idle waiting for work units to become available instead of doing useful work. A balance must be struck, however, when partitioning a model. If the work units contain too few LPs then the amount of work for that portion of the simulation may become trivial compared to the overhead of leasing the LP to a client. Conversely, if the work units are "too large" then the amount of state that must be transferred upon the completion of a work unit may be prohibitively large, and the number of available work units may be too small relative to the number of client machines.

With the addition of lookahead and work unit partitioning, we can build a model for total average application run time (T_a):

$$T_a = \rho \mu N_{wu} \qquad (2.3)$$

where ρ is the average event processing rate (e.g., wall clock seconds per event), μ is the average leased event density (e.g., number of events per leased execution window), and N_{wu} is the number of work units in the system. Equation (2.3) approximates the time in seconds a work unit spends performing the actual simulation computation.

The final component for total run time is request or idle time. This is the time an Aurora client spends waiting in a loop for the server to respond with a work unit available for the client. A conservative synchronization algorithm based on global LBTS values is assumed. This requires at least twice the number of work units as there are clients so that no client waits for an available work unit given that all clients have the same processing

speeds. This is due to the LBTS value not increasing until the last work unit for the current LBTS is successfully returned. Total request time can be formulated as follows:

$$\alpha = \frac{N_{wu}}{C} \tag{2.4}$$

$$\lambda = N_{wu}\left(\frac{S}{L}\right) \tag{2.5}$$

$$T_r = \begin{cases} T_a\lambda\left(C - N_{wu}\right) & \text{if } \alpha \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where α determines work unit to client (C) ratio in equation (2.4). This provides a quantifiable value whether clients must block for a work unit to become available. Next, the number of total lease windows, λ, for the entire simulation must is calculated through equation (2.5) where S denotes the total simulation time and L is the lookahead. In non-uniform lookahead simulations, the λ value would be the sum of all per work unit lease windows over the entire system. Finally, the model for total average request time (T_r) can be constructed where the average application work unit run time is multiplied by the total number of lease windows and the difference between the available client pool and number of work units. Assuming no other server overheads, there should be near zero request time when there are enough work units (e.g., $\alpha > 1$). One contributor to overhead which is not captured is the deferred wait time. Although equation (2.5) partially captures deferred wait due to work unit unavailability, there are additional non-deterministic factors which are not accounted for. Additionally, equation (2.6) holds only if the amount of computation performed per work unit is relatively equivalent.

Using equations (2.2), (2.3), and (2.6), we can construct an efficiency rating, E:

$$E = \frac{T_a}{T_a + T_o + T_r} \tag{2.7}$$

The efficiency rating gives an approximation of how well the Aurora system is utilizing the available client pool given all clients have the same processor speeds, for actual simulation computation and progress. A higher efficiency rating represents a properly partitioned simulation with relatively good computation to communication ratio and computationally intense work unit leases. This efficiency rating can be directly measured from performance data as the percentage of processor time spent in application code and is referred to as simply percentage application processor time in performance results.

2.6 Performance Study

The Aurora system provides a simulation infrastructure for applications to be deployed over a wide array of clients using different languages and machine architectures through the use of web services. Although interoperability is an important feature, the Aurora system attempts to deliver high performance services, consistent with interoperability goals. The web services based Aurora system was built using gSOAP 2.7.6c, compiled using gcc 3.4.3 with the -O2 optimization flag. The Aurora server was run in standalone mode. Machines designated as *Xeon* consist of two 2.8GHz Intel Xeon processors while *Pentium III* machines consist of eight 550MHz Intel Pentium III processors. Both machine types have 4GB memory using RedHat Linux connected with Fast Ethernet. The Xeon and Pentium III machines do not reside on the same LAN.

2.6.1 Microbenchmark Timings

The overhead for each web service method invocation (i.e., request-response pair) should be kept as small as possible. Because a master/worker PDES system requires work unit state and message buffers to be transferred between clients and server, transmitting binary data and the associated data transformations must be completed as efficiently as possible, even for large transmissions.

Table 3: Latency (ms) of Aurora Web Service Routines

	Messages with null (0 KB) payload
Request WorkUnit	0.563
Request State	0.535
Request InputMsgs	0.528
Return Results	0.613

Table 4: Data Transfer Time (ms) for Aurora Web Service Routines

	1 KB	**100 KB**	**1 MB**	**10 MB**	**100 MB**
Request State	0.772	14.196	129.451	1282.182	12814.652
Request InputMsgs	0.765	14.319	132.055	1308.921	13201.016
Return Results	0.939	26.679	262.503	2642.011	26638.825

Table 3 shows the total time in milliseconds to invoke a web service routine using null message sizes for *Request State*, *Request InputMsgs*, and *Return Results* using *Xeon* machines. Without the extra payload of state vector or message data, these timings show the minimum amount of overhead incurred per web service call. It can be seen that each

web service method exhibits under one millisecond latency when performed within a LAN environment.

XML and Base64 encoding of binary data is another source of overhead. As the size of binary data transformed for transmission over the network increases, so does the amount of overall time required for the web service method invocation. Table 4 shows the amount of time required for each web service call as the amount of data is increased. *Return Results* includes both state vector and output buffer transmission. Although gSOAP contains optimized Base64 routines and streaming XML techniques, the data transfer tests exhibit somewhat mediocre performance. The XML overheads prohibit maximum speed transfers, utilizing only approximately 60% of the available bandwidth throughout the variable payload size tests.

2.6.2 *PHOLD*: Synthetic Workload

The *PHOLD* application [90] was used to evaluate the performance of simulations running on the Aurora system using controlled workloads. The amount of computation performed by each execution of a work unit and the amount of state and message data transferred between clients and server were varied. In the following tests, a *linear PHOLD* model was used where generated messages are sent to an LP's immediate left and right neighbors. *PHOLD* was configured with 20 LPs, 500 KB of state, and lookahead of 1.0 seconds. The simulation was run over ten *Xeon* client machines with each client requesting 20 total work units.

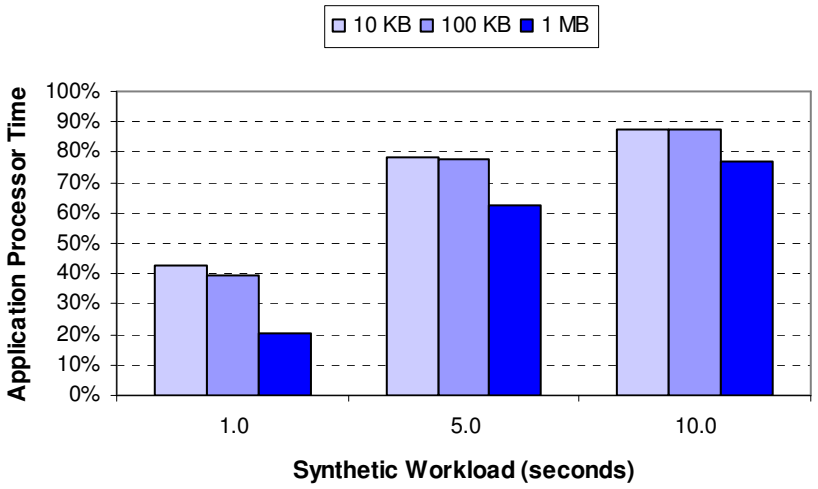

Figure 10: Effect of Workload on Performance

Figure 10 shows the fraction of processor time at the client devoted to performing computation for the actual simulation for each work unit leased to a client. The amount of data represented in each bar maps to the amount of message data transferred one-way in addition to the 500KB of state. The total amount of message data transferred per cycle would be two times the amount shown because messages must be downloaded from and uploaded to the server. As expected, as the synthetic workload per work unit is increased, the relative amount of time spent in overheads decreases. These results indicate that simulations should have a non-trivial amount of computation in order to achieve simulation application processor utilization that exceeds combined overheads.

2.6.3 Execution on Shared Resources

One of the goals of the Aurora system is to have the ability to run simulations in the "background" on hardware of varying speeds and architectures that are shared with other users. Coupled with the fault tolerance features being developed, the Aurora system, in principle, allows a simulation to in principle run virtually anywhere while

tolerating node failure and automatically balancing workload among the available processors. For example, the Aurora server can be on a dedicated machine while the client machines are run from idle desktop machines or unreserved public computing clusters.

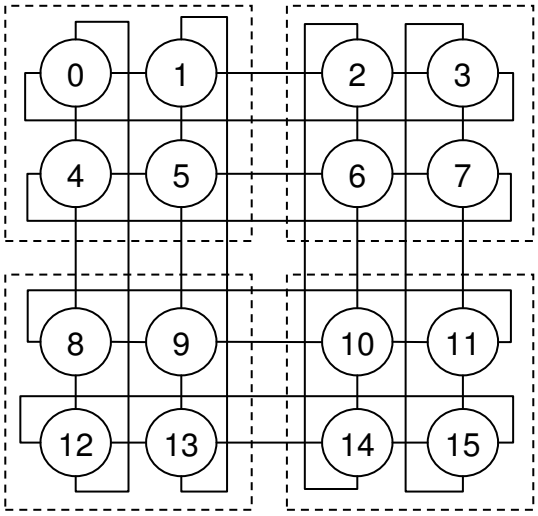

Figure 11: Example 4x4 Torus Queuing Network

A queuing network is used for this evaluation. The servers in the network are connected in a torus as shown in Figure 11. The dashed boxes designate the aggregated servers (subnet) mapped to a single work unit where each client would simulate a 2x2 torus network section. The links internal to each subnet can be different speeds compared to the links that connect subnets together.

The first experiments vary the delay on links between work units, thus varying the lookahead. The queuing network is configured as a 250,000 server 500x500 closed torus network partitioned into 625 20x20 torus subnets which can be leased as work units. The internal links within each work unit are set at a delay of 10 microseconds. The job

generator creates 10,000 local jobs with random server destinations that exist within the work unit and an additional 10,000 remote jobs with random server destinations that are external to the work unit. Jobs reaching their destination server are assigned a new random destination according to their previous local or remote designation to keep the relative amount of local and remote jobs consistent. The service time for jobs is exponentially distributed with a mean of 5 microseconds. Generated and subsequent processed jobs that have timestamps greater than the leased window end time are not be processed during the current lease. These messages would be packed into the output buffer and sent back to the server for future execution.

There are a total of 78 clients, consisting of 14 *Xeon* processors and 64 *Pentium III* processors. The machines are unreserved and some are heavily loaded (e.g., 80-100% load). No special nice priorities are given to the Aurora clients. The Aurora server was run from a dedicated *Xeon* machine.

For the following figures, the overhead time includes state vector and input (server to client) or output (client to server) buffer data transmission and other various overheads such as server-side message binning and client-side input message queuing. The request or idle time is the amount of time the client spends in the handshaking/authorization/work unit request loop. If no work unit is available, the client sleeps for 1 second and tries again in order to avoid flooding the server with work unit requests. All times are averages across all clients.

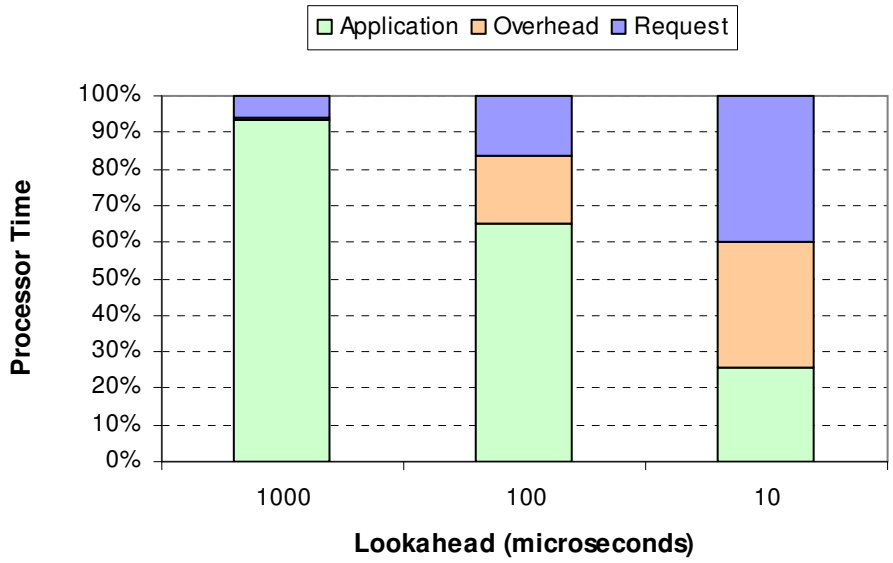

Figure 12: Effect of Lookahead on Performance

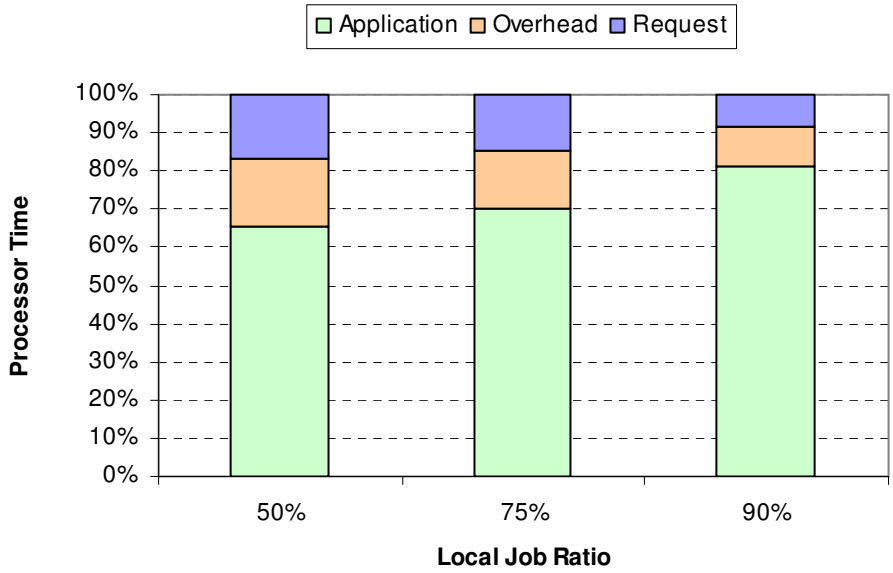

Figure 13: Effect of Relative Workload on Performance

The first scenario involves varying the delay on links between work units, thus varying the lookahead. Figure 12 shows the processor time used for running the simulation, overhead time, and request time. As lookahead decreases the number of

messages that can be processed during the work unit lease decreases although the number of messages generated per work unit lease remains constant. Consequently, more messages will exceed the leased simulation end time thereby increasing the number of output messages that must be buffered and sent to the server.

The next test modifies the percentage of jobs destined for servers local to the work unit. The delay between work units is held constant at 100 microseconds. As the relative number of jobs destined for local servers increase, the amount of processor time dedicated to the actual computation increases as well. Figure 13 shows increased computation at the clients resulting in less data transmission thereby reducing server load and overhead time.

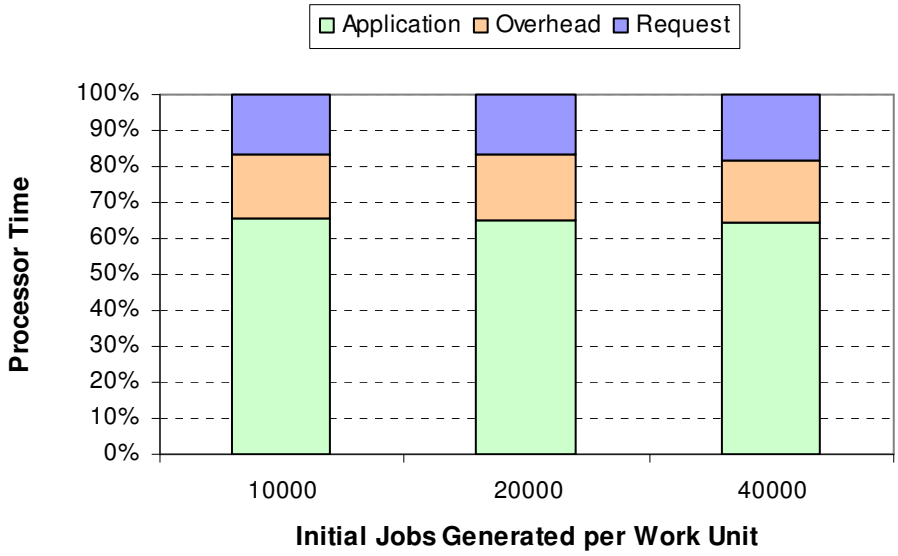

Figure 14: Effect of Absolute Workload on Performance

Figure 14 illustrates the impact of modifying the absolute workload per work unit. The delay between work units is kept constant at 100 microseconds with 50% local and 50% remote jobs. Efficiency remains relatively constant as the number of jobs in the

system increases. Although the raw overhead and request times increase for each case, the application run time increases as well due to the increased workload from doubling the number of jobs in the system. These experimental results suggest a proportional increase in each of the three areas contributing to the total execution time. It would be expected that this trend would hold constant until bandwidth or memory becomes an issue.

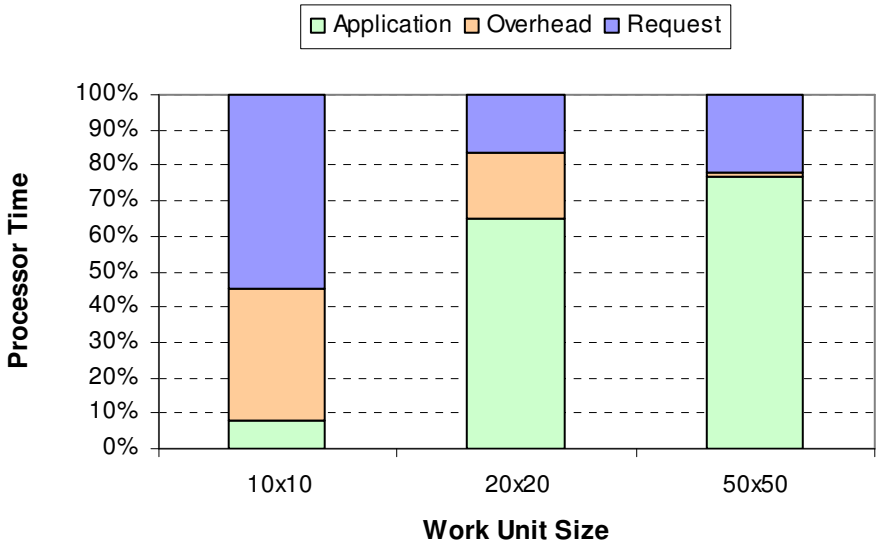

Figure 15: Effect of Work Unit Size on Performance

The final test evaluates the impact of varying the number of partitions of the torus network. The link delay between work units is 100 microseconds across all tests. The number of jobs generated is set at 10,000 local and remote server destinations. The torus network is partitioned into toroidal subnets of size 10x10 (2500 work units), 20x20 (625 work units), and 50x50 (100 work units).

As partition sizes increase, the number of local links a remote job must traverse increases significantly. Since the internal links between servers are 10 microseconds, a remote job may potentially traverse up to approximately 9 times within a work unit before either the LBTS limit is reached or the job reaches the partition boundary. Figure 15 shows that as the job route length to reach a partition boundary increases as partition size increases, the amount of local computation for larger partition sizes increases.

The large percentage of overhead and request times for the 10x10 subnet case can be attributed to relatively little computation performed at the clients compared to the time taken to transfer data. There is more server contention for smaller partition sizes as the work unit return rate is higher for smaller partition sizes increasing server load, thus, the decreasing *Overhead Time* as the partition sizes increase as shown in Table 5. Increased server load contributes to increased request time as this monolithic web services based master/worker PDES system is not multi-threaded or distributed.

Table 5: Total Average Overhead and Request Times (sec)

	10x10	**20x20**	**50x50**
Overhead Time	1036.396	32.710	14.474
Request Time	1528.394	30.055	210.766

The 20x20 subnet case shows a reduced percentage of overhead and request times compared to the 10x10 case as there is more computation performed at the clients. The 50x50 subnet size does not follow a decreasing *Request Time* trend. Due to the conservative synchronization method used in the current implementation of Aurora, there must be approximately twice the number of work units as there are clients. Under

conservative synchronization, the simulation application must be partitioned to accommodate enough work units for the anticipated number of clients to avoid the increased *Request Time* shown in the 50x50 case.

2.6.4 Hybrid Shock Discrete Event Simulation

For this test, a particle physics simulation was used. This simulation models shockwave propagation using electromagnetic hybrid algorithms with fluid electrons and kinetic ions [91]. The simulation space is partitioned into cells, each containing an initial number of ions. Ions move from one cell to another in accordance with the electromagnetic forces acting upon it. This simulation relies on a lookahead (LA) parameter. In addition to its usual meaning, the LA parameter provides a means to avoid retracting previously scheduled events. Specifically, the time at which a particle is expected to move from one cell to another, i.e., its "move time" is estimated, but the move event is not scheduled unless it has a timestamp in the interval [current time + LA, current time + 2*LA]. Move times that are far into the future may need to be recomputed later in the simulation because of subsequent changes in the electromagnetic forces. Acceptable LA times were determined experimentally, with smaller values increasing accuracy, but at the cost of reduced parallel performance. This simulation generates three types of events: an ion move event when aforementioned conditions are met, an update notification event when a field update occurs, and a schedule next ion movement event for processing ions. This simulation aggregates cells into work units which can be leased to clients. For this performance test, there were 20 LPs of 100 cells each, where ten Xeon clients were used. The lookahead is kept constant at 0.11 for these tests. The amount of

state is non-trivial as each cell and ion requires 180 and 140 bytes of memory respectively to save.

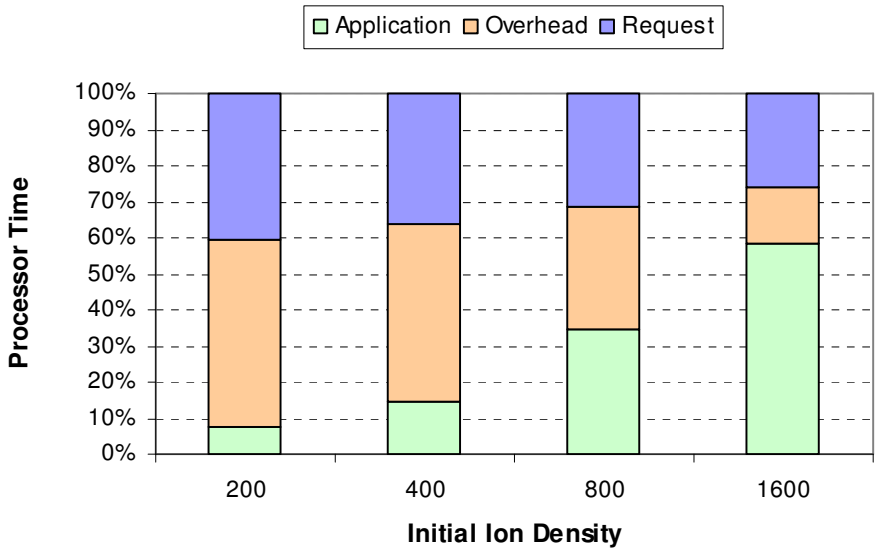

Figure 16: Effect of Ion Density on Performance

The cell width is held constant while the number of initial ions was varied from 200 to 1600 per cell. Figure 16 shows that if event computation per cell is sparse, the relative amount of processor time dedicated for simulation progress is low. Request and overhead times accounted for a large percentage of the total overhead as the single-threaded, single-process server must block for large amounts of incoming and outgoing state. Even with the limitations in the web services based Aurora server implementation, performance of this real-world application illustrates that Aurora can be an acceptable platform for PDES; however, the master/worker paradigm is not suitable for all applications, such as simulations with low concurrency.

2.7 Conclusion

The web services based Aurora system provides a new approach for PDES by utilizing the master/worker paradigm and leveraging the interoperability of open standards. The Aurora system delivers an application-independent simulation framework that can be run using various languages and on a variety of different hardware architectures delivering adequate performance under a variety of conditions. The Aurora system is extensible, and can be expanded to enhance performance, load balancing, and security. We demonstrated Aurora's strength with running simulations in the "background." The Aurora system affords simulation application developers the ability to create and run simulations without having to worry about client failures, varying machine architectures, processor speeds, and load on unreserved machines.

The performance study shows that even under a web services framework, a master/worker PDES system can provide a portable platform for certain PDES applications. We have shown that conservatively synchronized PDES codes that exhibit good lookahead with computationally intense work units can attain high efficiency ratings (relative amount of processor time spent in application code). A torus queuing network with the proper amount of work units available in the worker pool with favorable parameters typically exceed 50% application processor time. Although utilizing only half of the processor time is most likely less efficient than a comparable conventional PDES system, these resources that compose the worker pool are either work sharing or idle, in which case the cycles would have been wasted if not utilized by this system. Under non-synthetic workloads, the Hybrid Shock application showed high efficiency with a large amount of interacting ions per cell or high ion densities. These results enforce the idea

that master/worker PDES systems are most conducive to computationally intense simulations.

A metacomputing master/worker PDES system is not suitable for every simulation. Fine-grained simulations that do not exhibit good concurrency and lookahead are better suited for conventional PDES executions. There are limitations specific to a monolithic design around web services, such as performance degradation due to XML overheads, a non-distributed centralized architecture, and prohibitive infrastructure for running multiple concurrent simulations under a single master instance with compile-time bindings to specific application code.

Although web services provide a platform for complete language independence through widely portable standards, the aforementioned issues can prove to be problematic with general acceptance of such a vastly different framework for PDES than traditional systems. An alternate approach to master/worker PDES is to utilize highly portable libraries with language specific bindings, but sacrifice little in platform portability. In addition, such a system will offer higher throughput and performance by eliminating overhead culprits such as XML. A distributed approach to master/worker PDES will allow a scalable system, allowing concurrent large-scale PDES codes to be run simultaneously without the restrictions of a singular application-bound master service.

CHAPTER 3

A CONCURRENT, DISTRIBUTED APPROACH TO MASTER/WORKER PARALLEL DISCRETE EVENT SIMULATION

The master/worker system based on web services to deliver high throughput PDES performance using a computational infrastructure is vastly different from the paradigm typically used for PDES codes. Although the feasibility of certain PDES applications on this master/worker infrastructure was examined, large-scale simulations suffered performance penalties inherent in the architecture. This was a major issue as often the decision to parallelize a discrete event simulation is to gain speedup over a serial implementation. Therefore, an alternate approach to system design was explored, where a small sacrifice in portability is exchanged for significant design changes to improve the performance of large scale simulations. While performance under the first system was acceptable for test cases exhibiting good lookahead and computationally intense work units, the single server design coupled with large-scale simulations would undoubtedly incur too much overhead thus diminishing throughput gains achieved by capturing cycles on idle desktops.

From the performance data exploring a web services based approach, some conclusions and observations can be drawn:

1. Although a master/worker system by design requires a master server, the master service behaved in *three distinct* ways: metadata management, state vector storage, message processing. The necessity of strictly adhering to a centralized master

server is not required. Although multiple servers were used for fault tolerance, these additional servers were intended exclusively for fault recovery through replication and not to enhance performance.

2. The performance of a widely portable master/worker PDES system is severely degraded under a web services framework due to XML-associated encoding and processing overheads. Although some performance loss was expected, the amount of data that is transferred between workers and the master service is not typical for traditional web services use and thus exacerbate overheads. State vectors and messages tend to be large which is problematic with XML encoding.
3. The absence of thread-level concurrency has tremendous effects on serializing the computation on *both* the master and workers. It is apparent that multi-threading is not an optional component, but is required throughout the entire system for high performance.
4. A master/worker PDES system based on web services can be cumbersome to use due to per-application stub generation and application-specific WSDL definitions on both the server and the client. Although this can be partially avoided using a more generic approach to service design, it is still necessary to devise an alternate method to provide support for both concurrent and replicated simulations.

With these limitations to the first generation Aurora system, it was apparent that significant changes to the core architecture were needed to accommodate large-scale simulations with a larger worker pool. The single server design of the first generation Aurora system could be modified to accommodate extra load by multi-threading and/or

expanding the Aurora server-side services, however, the web services base hindered the ultimate driving design goal of scalable high performance PDES execution. This revision of the system architecture allowed for the elimination of key overheads as well as broadening support for PDES codes.

The conclusions drawn from experiences with a web services based master/worker PDES offer a concrete assessment of system behavior and shortcomings with regard to performance. The next section explores more master/worker PDES limitations in detail to provide an overview of issues with this style of computation to help better formulate alternate approaches for a more balanced and higher performing master/worker PDES infrastructure.

3.1 Master/Worker PDES Issues and Limitations

Although nearly all PDES applications can be adapted for a master/worker paradigm, there are certain issues and limitations. First, all work units are subject to migration due to the dynamic nature of the client worker pool. Consequently, the system must store simulation state such that restoration of state vectors at a later time is possible. True transparency in state saving with minimal impact to the application programmer is accomplished by saving the contents of the program's stack, file handles, shared library code, etc. as done in Condor [74]. This, however, limits the re-execution of the work unit to clients with matching operating systems. This presents problems for master/worker PDES on two levels. First, a pool may contain only a limited number of nodes of similar architecture for which a work unit was assigned. If these nodes suddenly are no longer available, the simulation will be blocked, with no means to rollback the simulation to a

previous state because once a work unit is returned to the master, guarantees about advancing simulation times can be made. Second, adhering to true architecture and operating system independence allows public resource computing infrastructures to provide the widest flexibility in client worker pools. As a result, master/worker systems are limited to architecture and operating system agnostic packing and serialization protocols. This results in routines that are directed by the application programmer using methods and functions provided by the simulation engine API.

Similar to the issue of state vector migration, all messages generated must be serialized in an architecture and operating system independent manner, as a message generated on one client may be destined for a work unit that may be hosted on a dissimilar machine. This is unavoidable overhead incurred for every generated message. PDES codes that generate excessive numbers of messages may be less suitable for a master/worker PDES system due to the additional memory requirements of message buffering and increased processor time consumed for processing.

Finally, in contrast to traditional peer-to-peer PDES systems, master/worker systems must store simulation data in a centralized fashion. Although the master is logically centralized, it does not mean it must adhere to a centralized design physically. Distribution of functionality across multiple machines can lead to better load distribution. Moreover, these machines must be high performance with accompanying large bandwidth links for the amount of incoming and outgoing aggregate data. Master services that store state or message data must have fast memory subsystems with large memory pools for storing packed state vectors and buffering messages generated by work units.

The work presented in this chapter attempts to address problems and issues arising from a web services based approach along with limitations described above. Not all limitations can be eliminated, but some performance can be recovered by identifying areas that are prone to degrading into serial execution and eliminating unnecessary overheads introduced by outside sources such as XML.

3.2 Addressing the Issues through a Concurrent, Scalable Design Solution

The Aurora2 architecture is a departure from the first generation design based on web services in the following important areas: distribution of the master server into multiple functionally different multi-threaded services, a less overhead-prone communication middleware, and concurrent simulation and simulation replication support (i.e., performing a PDES execution multiple times for data and output, not for fault tolerance).

The area that received the greatest performance improvement was the change from a single master server design to multiple functionally different multi-threaded servers that could be dynamically allocated. As discussed earlier, although the master is required to handle all logistics with simulation metadata, state vector storage, and message storage and forwarding, all of these tasks can largely be performed independently of each other. Moreover, the two main contributors to overhead from work units come from state vector storage and message buffers whose overhead is proportional to the amount of state and emitted messages. The functionality of the master service was separated into a lightweight master controller, a work unit storage service, and a message storage service that collectively comprise the master back-end services.

This allows dynamic deployment of back-end services tailored for a particular simulation or for large-scale simulations in general although logically, the master service functions as a single entity. Additionally, the client software was re-designed to be multi-threaded. It was not entirely apparent that multi-threading the client was necessary during the design of first web services based master/worker PDES system. However, it was observed through simulations that exhibited large states or generated a large number of messages, that the inability to concurrently package and send these updates over the network significantly serialized the computation, reducing the benefits of a master/worker approach to PDES.

The web services communication framework in the first generation Aurora system was replaced with a traditional sockets-based framework allowing for higher performance data transfers that account for a non-trivial portion of the overall overhead. Serialization of data to a compact binary format offers improved throughput rates and reduced processor load while processing messages.

One of the important uses of a high throughput simulation system is to not only run a distributed simulation across many processors efficiently but also to create replicated runs of large-scale simulations. The elimination of compile-time bindings and strict web services compliance allows for the support for simultaneous simulations using only one set of master services.

To further define the space for which these issues and accompanying solutions exist, PDES application properties for which a master/worker system employing conservative time management are targeted for are characterized concretely next.

3.3 Desired Characteristics of Conservatively Synchronized Simulations in a Master/Worker System

Tightly-coupled cluster and supercomputing systems will always offer maximal speedup for PDES codes. The master/worker PDES infrastructure allows an alternative method for expanding processing capabilities of a site through the re-use of existing infrastructure and idle workstations. Although almost all PDES codes could in principle run on a master/worker system, the limitations of the infrastructure can outweigh the benefits.

Conservatively synchronized PDES codes exhibiting the following properties can benefit the most from a master/worker infrastructure:

1. Favorable lookahead and predictability
2. Computationally intense work units
3. Favorable computation to communication ratio
4. Compact state vectors with low memory footprints

The main weaknesses in a master/worker PDES system are the potentially lower bandwidth links between workers and the master services along with indirect messaging and buffering on the master. In order to mitigate these disadvantages, the PDES application should exhibit the aforementioned qualities.

Lookahead, or predictability of a simulation, must be favorable to allow sufficient computation to occur during the work unit lease. Smaller lookaheads relative to the available simulation time for computation per work unit lease increases synchronization costs and work unit return rate, decreasing the concurrency of the system and increasing the overall time spent in overheads.

Moreover, favorable lookaheads may result in significant portions of simulation time to be leased as work units, however, if the actual PDES application itself performs few computations during the work unit lease the amount of time spent in actual real work for forward progress will be overshadowed by the synchronization and work unit lease and return costs. PDES codes in a master/worker system must be computationally intense in order to take advantage of the available processing power in the worker pool without overwhelming overheads. Computationally intense workloads are codes that encompass non-trivial computations requiring many processor cycles to achieve a desired result. For instance, a simple Monte Carlo simulation may consume significantly less processor time than PDE solvers employing FFTs. This property is important for sustaining concurrency within master/worker PDES system.

As with all distributed computing programs a favorable computation to communication ratio (e.g., relative wall clock time spent in application computation versus communication) is necessary to allow speedup over sequential implementations. Communication time includes all time spent in lease and update protocols between the master and workers including retrieving lease information, state vector data and input messages. Additionally, this includes time to update state vectors and any output messages that are generated. The time spent during these operations is considered overhead because they are not part of the processor time dedicated to forward simulation progress (e.g., wall clock time spent in application code). Even with generous lookahead and computationally intense work units, if the amount of data that must be shifted between the master service and clients is large due to the state vector or message data (e.g., number of messages generated or a large average payload per message), this will

lead to excessive consumption of bandwidth and increased transmission time ultimately decreasing the parallel efficiency of the simulation.

Similarly, a simulation with compact state is the best match for master/worker PDES because it avoids consuming large amounts of processor time and communication bandwidth for state synchronization updates and state leases to available workers. State vectors must be unpacked at the beginning of a lease to a client after retrieval and packed and updated after the lease ends to a client.

These application properties are crucial for providing efficient execution in a master/worker PDES architecture and the veracity of these desired properties will be quantified in the performance study.

3.4 Aurora2: A Scalable, Distributed Architecture for Master/Worker PDES

Aurora2 is an approach, through the adherence to master/worker design principles, to provide a distributed system across loosely coupled machines of varying operating systems and architectures providing computational capacity and throughput for large-scale parallel and distributed simulations. This approach is a departure from the previous web services based design with enhanced scalability, application insulation, and concurrent simulation support.

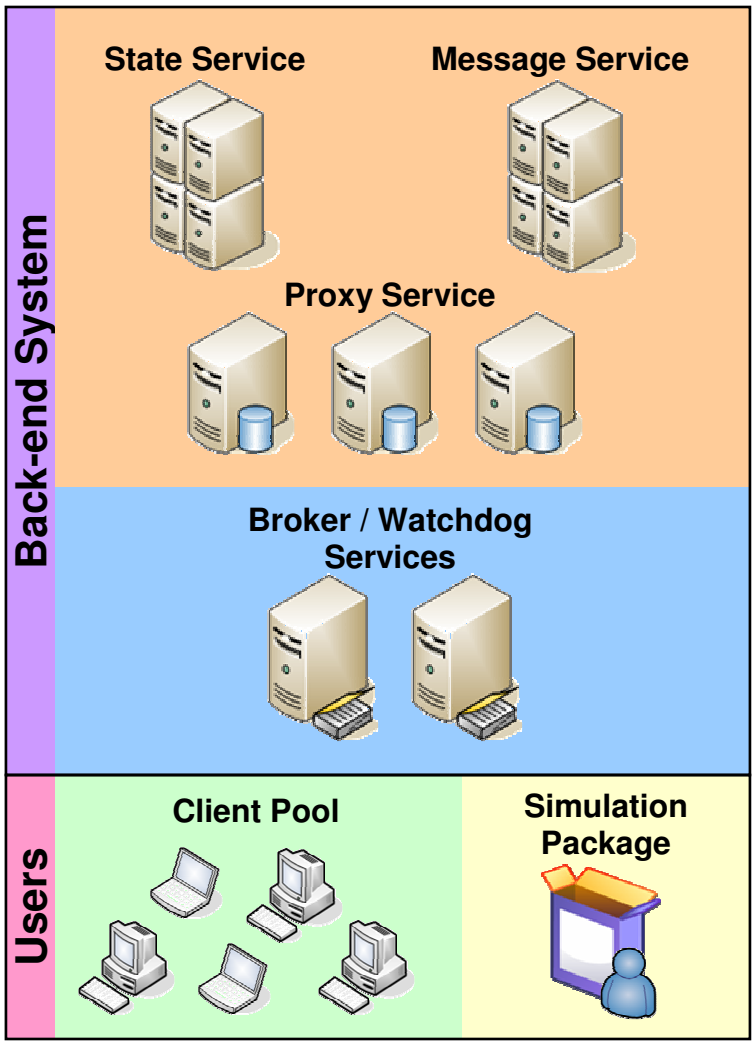

Figure 17: Overview of Logical Aurora2 Components

The Aurora2 system can be logically divided into three parts: master back-end services that include state, message, proxy, and broker services, simulation packages, and the client pool that provides computational power, as shown in Figure 17. For master/worker PDES systems, we assume that the services and clients exist on the same side of a firewall or that steps are taken to allow communications to such services that span a firewall such as open ports or SSH tunneling. The work unit pull-based system

does not require clients to act as servers. Thus client requests are unidirectional in nature (e.g., client to server only).

Although complete independence from architecture and language binding is sacrificed because of the removal of the web services base, this approach remains highly portable due to the foundation library used as the underlying framework for all Aurora2 services. The Portable Components (POCO) for C++ library is freely available, open source solution providing standardized abstractions for network-centric applications [92]. POCO provides clean and portable interfaces filesystems, networking, threads, etc. As Aurora2 is built upon this library, the architecture can be compiled on any operating system that can compile POCO, which encompasses most modern operating systems including, but not limited to: Windows, Linux, Mac OS X, Solaris, HP-UX, Tru64, OpenVMS, and embedded systems such as Windows CE and QNX Neutrino.

3.4.1 Broker Service

The broker service acts as resource directory for all services and clients within the system. The broker service provides information such as the current master proxy host address or unique key identifiers to host addresses for work unit state and message services to the clients. The broker performs automatic heartbeats to the proxy service and election of new proxies should the master proxy crash. This information is then queried by clients or other back-end services after a connection timeout to the proxy service that has crashed or is no longer reachable on the network. Optional watchdog services can provide restart and monitoring functionality for other back-end components including the broker.

3.4.2 Proxy Service

The proxy service is the central core controller and is regarded as the logical master in the master/worker paradigm. Internally, major metadata portions are contained within managers as shown in Figure 18. The proxy contains metadata for all simulation packages and oversees the other two back-end components: the state server and message server. Incoming connections are accepted by the connection manager thread that services the network. Valid connections are then passed to a thread in the generic worker thread pool that is loaded with data specific to the request and then is serviced by invoking applicable methods in one of the metadata managers: simulation package manager, work unit server manager, message server manager, or client manager.

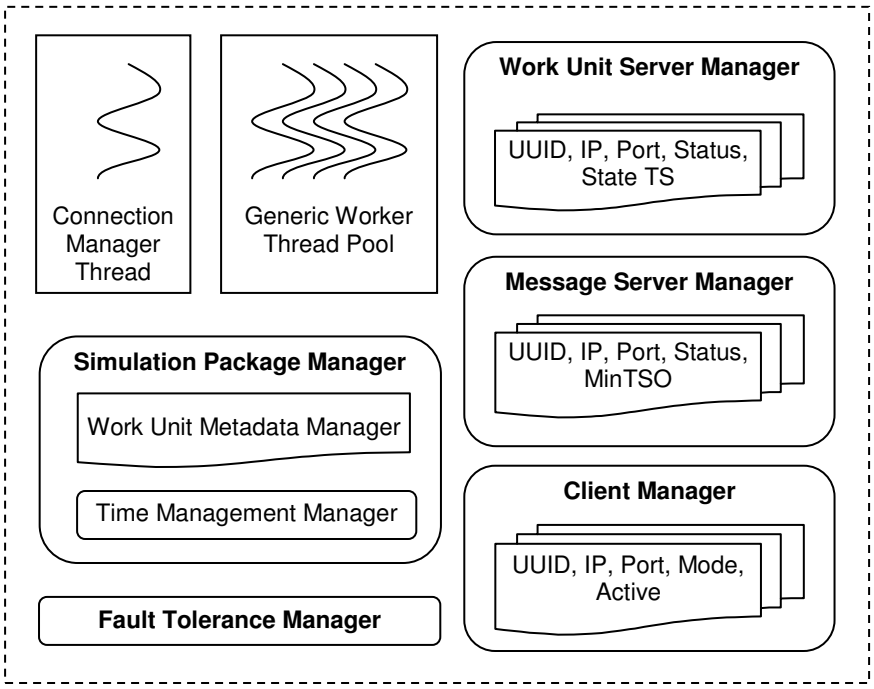

Figure 18: Components of the Proxy Service

The simulation package manager contains the work unit metadata manager and the time management module. Information pertaining specifically to the simulation work units such as simulation times, consistency of state vectors, and reverse lookup tables to work unit state and message stores are contained in the work unit metadata. The time management manager is instanced within the simulation package manager that controls global simulated time. This manager contains aforementioned methods to compute safe processing bounds.

The time management manager can include a mixture of conservative (blocking) or optimistic (rollback-based) schemes. The time management mechanism can be chosen at runtime by the simulation package definition. Multiple different time management schemes can be used for separate simulations all running on the same back-end instance. The current version of Aurora2 includes a centralized conservative time management mechanism with an identical algorithm implementation as discussed previously. If the system were to allow multiple proxies through a front-end load balancer, distributed time management would need to be addressed.

The client manager stores information about the client such as IP address, global unique client keys, and any outstanding work unit leases. The other two managers inside the proxy service handle metadata for the ephemeral data stores involving work units. The work unit server manager oversees work unit server instances storing information such as server IP address, port number, memory allocation, and listing of simulation packages the server is hosting. The message server manager stores similar information but is instead dedicated to handling message server specific metadata.

3.4.3 State and Message Services

Due to PDES applications inherently belonging to a different class of problems typically associated with distributed computing programs on volunteer computing and desktop grid architectures, it is necessary to add two additional storage services. State vectors are modified as the work unit advances in simulation time. Unlike typical EP classes of problems, the simulation time must be limited so that a work unit does not pass an upper safe processing bound (e.g., MinETS) to preserve the LCC. When the simulation time reaches this end time limit, the final state vectors and any messages generated during the simulated time must be stored for a future work unit lease to another client.

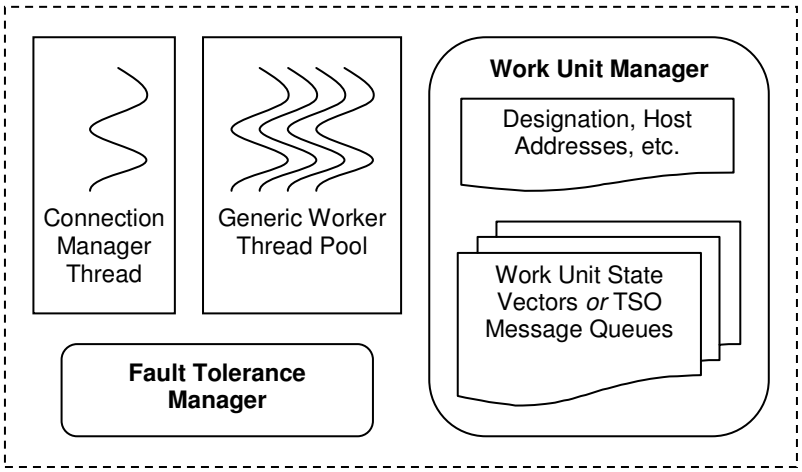

Figure 19: Components of Storage Services

The state and message services provide a distributed storage system for collecting pertinent states. Due to the possibility of a high volume of clients that may upload large state vectors and messages, these services must be designed for high performance. Following a similar design as the proxy service, these state services are multi-threaded

and any number of them can be instanced for improved scalability as shown in Figure 19. Depending upon the functionality of the service, the work unit manager stores either state vectors or a TSO message queue. State vectors are stored as application-defined contiguous blocks of memory that are packed and unpacked by simulation-specific routines overwritten by the Aurora2 clients. Similarly, packaged messages are stored in their respective destination LP TSO message queue.

3.4.4 Client

The back-end services provide the necessary infrastructure to run PDES applications in a distributed fashion over loosely coupled resources. However, the clients perform the actual simulation computation. After communicating with the proxy service, these clients contact the proper back-end state services to download simulation state and associated messages per work unit lease. Computation is done locally and independent of other clients. Once the designated client simulation period is processed, the final state and messages are uploaded to the work unit state and message state servers. This process can repeat for a specified number of times, until the simulation end time is reached, or if interrupted by a user.

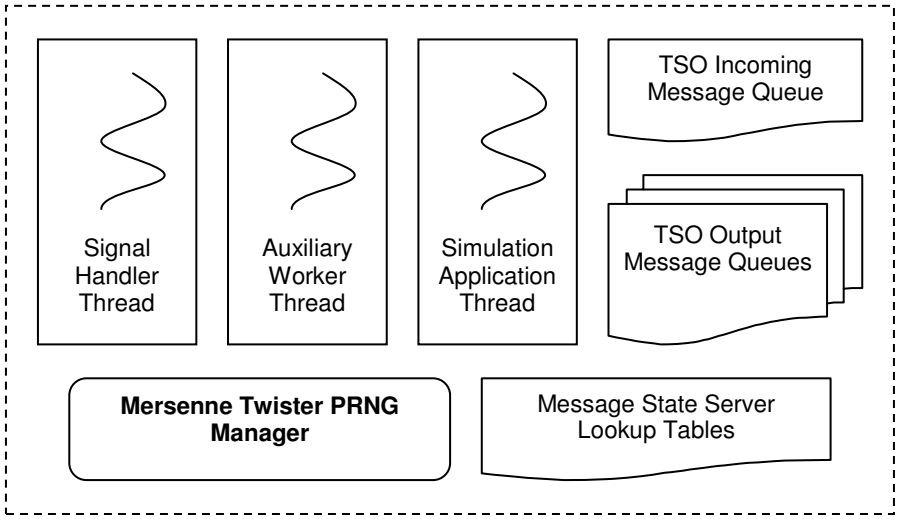

Figure 20: Components of the Client

The Aurora2 client is multi-threaded to allow unimpeded execution of the simulation application code while master/worker-specific tasks can run concurrently as illustrated in Figure 20. The auxiliary worker thread can perform background processing tasks such as message state server location lookups while the simulation is running.

The auxiliary thread is particularly important with regard to inter-LP messaging. When a message is generated during traditional PDES executions, the message is usually immediately sent to the recipient LP. In a master/worker metacomputing environment, connectivity between clients is not assumed; therefore generated messages are buffered at the client in the TSO output message queues. TSO message queues are priority queues where the order of messages is determined by timestamp and secondary tie-breaking fields of the message. These TSO message queues are arranged in increasing timestamp order. Due to the distributed nature of the message state service, lookups must be performed to ascertain the location of destination LPs (work units). This information is

gathered into the message state server lookup tables which are necessary for the work unit finalization task that runs after the simulation computation completes.

In order to ensure repeatable executions, the client contains a random number generator manager. The Aurora2 clients utilize the Mersenne Twister pseudo-random number generator [93]. Interfaces are automatically provided by the Aurora2 client for the PDES application to pull random numbers from uniform and normal distributions and can be seeded by the application as needed. Multiple parallel pseudo-random number generator streams are also supported by the client and can be automatically saved along with that of the default generator. This master/worker PDES system will generate identical results regardless of the type of time management selected if the same random number generator and seeds are chosen between runs.

3.4.5 Simulation Packages

With the addition of concurrent simulation support in Aurora2, a specification was needed to create a simulation instance on the back-end services. The simulation package definition accomplishes this by providing initial metadata about the distributed simulation such as the number of work units, a lookahead connectivity graph, simulation begin and end times, and deadlines in wall clock time for each work unit lease. In addition to these initial runtime parameters the simulation package definition may upload any initial work unit states. After the simulation package definition is uploaded to the proxy service, the necessary metadata tables and allocation of resources is done prior to client execution such as the distribution of work unit storage load. Initial work unit distribution, dynamic load balancing, and memory distribution and re-distribution

techniques are possibilities for efficiently distributing back-end load and are discussed further in chapter 6.

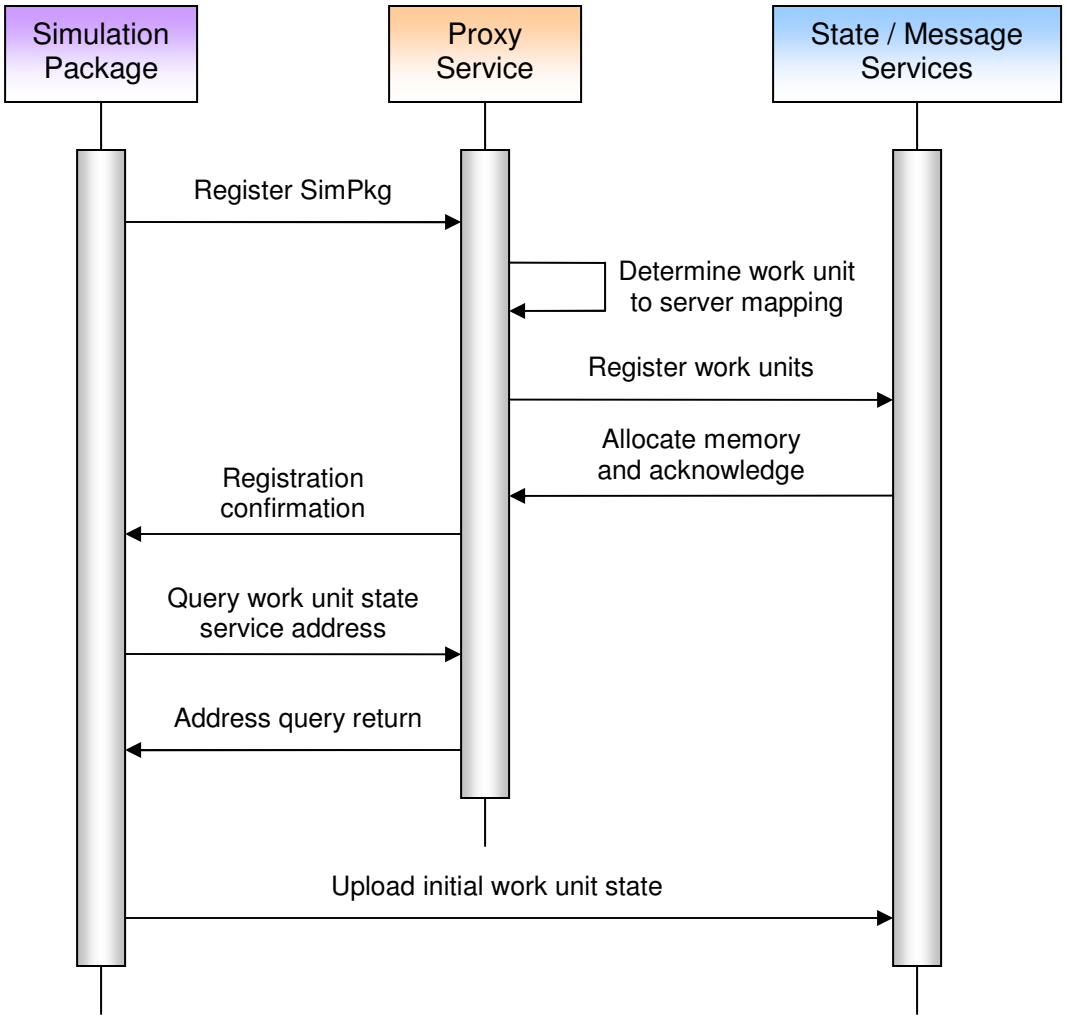

Figure 21. Simulation Package Interaction

Figure 21 illustrates the procedure for registering a simulation package with the back-end master services. After the simulation package definition is uploaded to the proxy service, the necessary metadata tables and allocation of resources is performed prior to client execution such as the distribution of work unit storage load. For example, Figure 22 illustrates uploading the simulation package definition to the proxy service

(step 1) where the simulation contains fifty total work units with an anticipated memory requirement of two MB per work unit in both state vector and message storage. The proxy service will then query its internal tables for available work unit and message state servers. 25 work units are allocated to each in steps 2 and 3 to work unit state servers 1 and 2 respectively. For message state storage, 37 work units are allocated in step 4 and the remaining 13 are allocated in step 5.

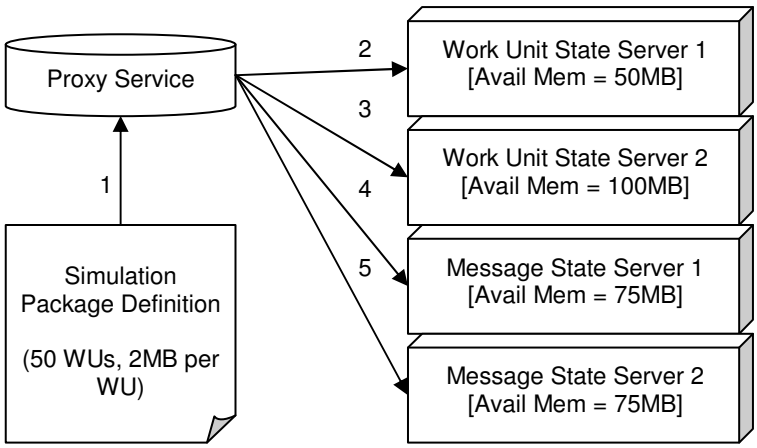

Figure 22: Example Work Unit Distribution

After the simulation package is registered with the proxy service, the simulation package can be specified to optionally upload initial work unit states if necessary. This involves querying the proxy service for the state service host addresses that the simulation package uses to directly contact and upload state.

3.4.6 Work Unit Lifecycle

The Aurora2 client performs a series of steps to download pertinent metadata, un-packaging of various data, simulation execution, re-packaging of state variables and messages, and finally uploading results back to the proper back-end services. These steps

can be characterized as the master/worker PDES work unit lifecycle. The lifecycle is comprised of three major steps, excluding the one-time initialization phase. This initialization phase includes thread initialization, signal handler setup, communications manager handshaking, and command-line parsing. The three major cyclic steps for a master/worker PDES client are: unit request and download, simulation application computation, and work unit finalization and upload. The client may fail at any point during these steps. A work unit that is not returned within its wallclock deadline lease period due to client failure is rolled back and leased to a different client. These steps are similar to conventional volunteer computing lifecycles. However, additional PDES requirements modify the details significantly. A general overview of the work unit lifecycle is presented in Figure 23 with an in-depth interaction diagram shown in Figure 24.

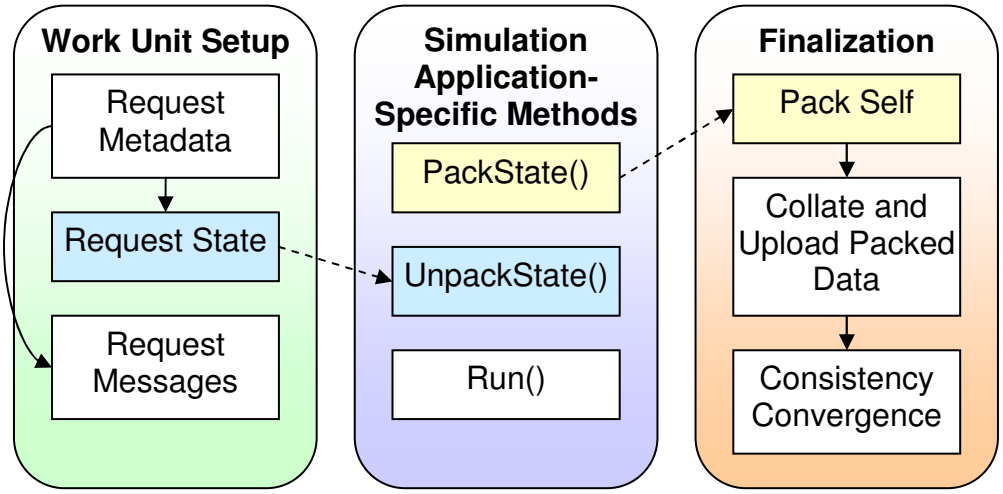

Figure 23: Work Unit Lifecycle

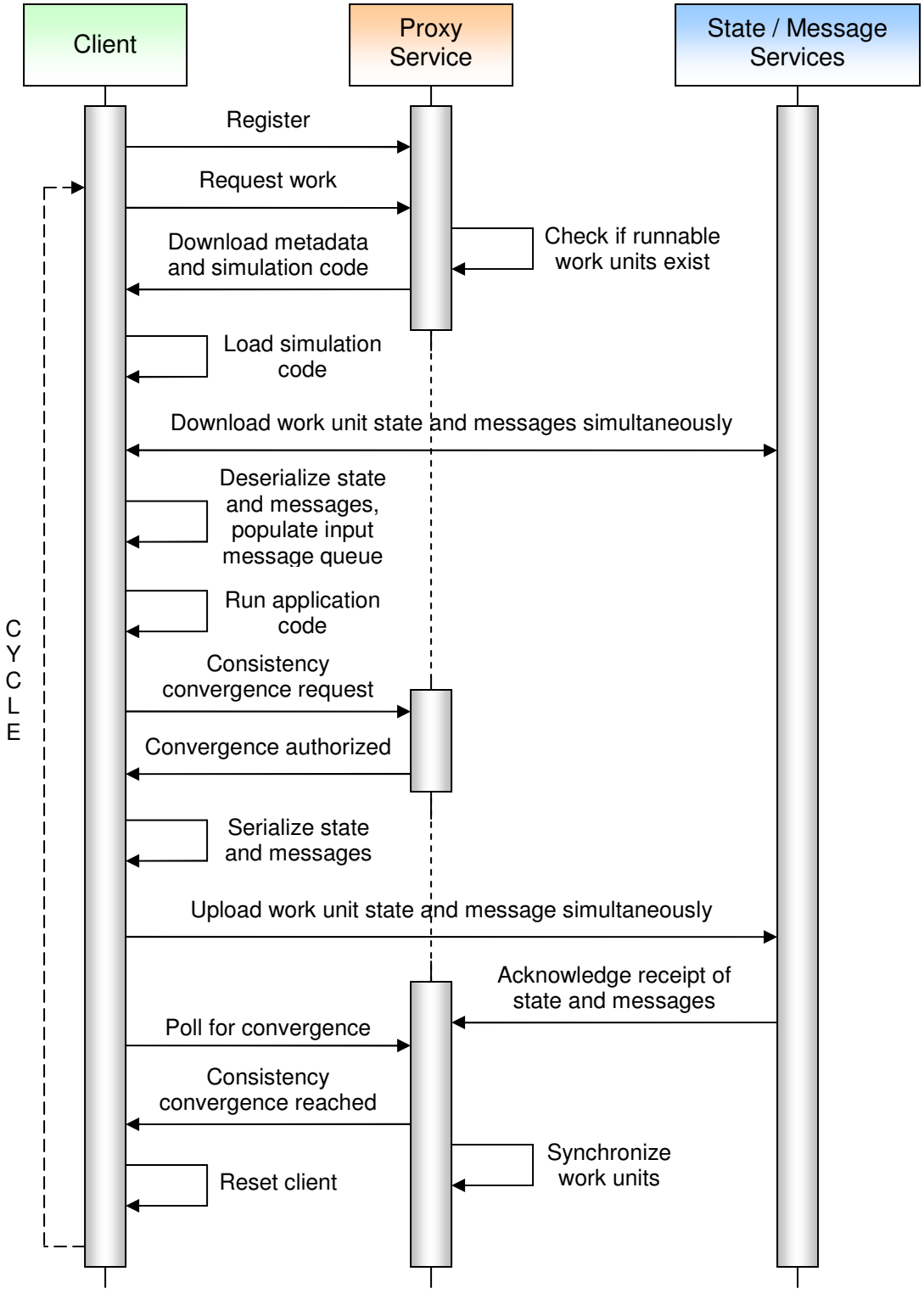

Figure 24. Client and Work Unit Lifecycle Interaction Diagram

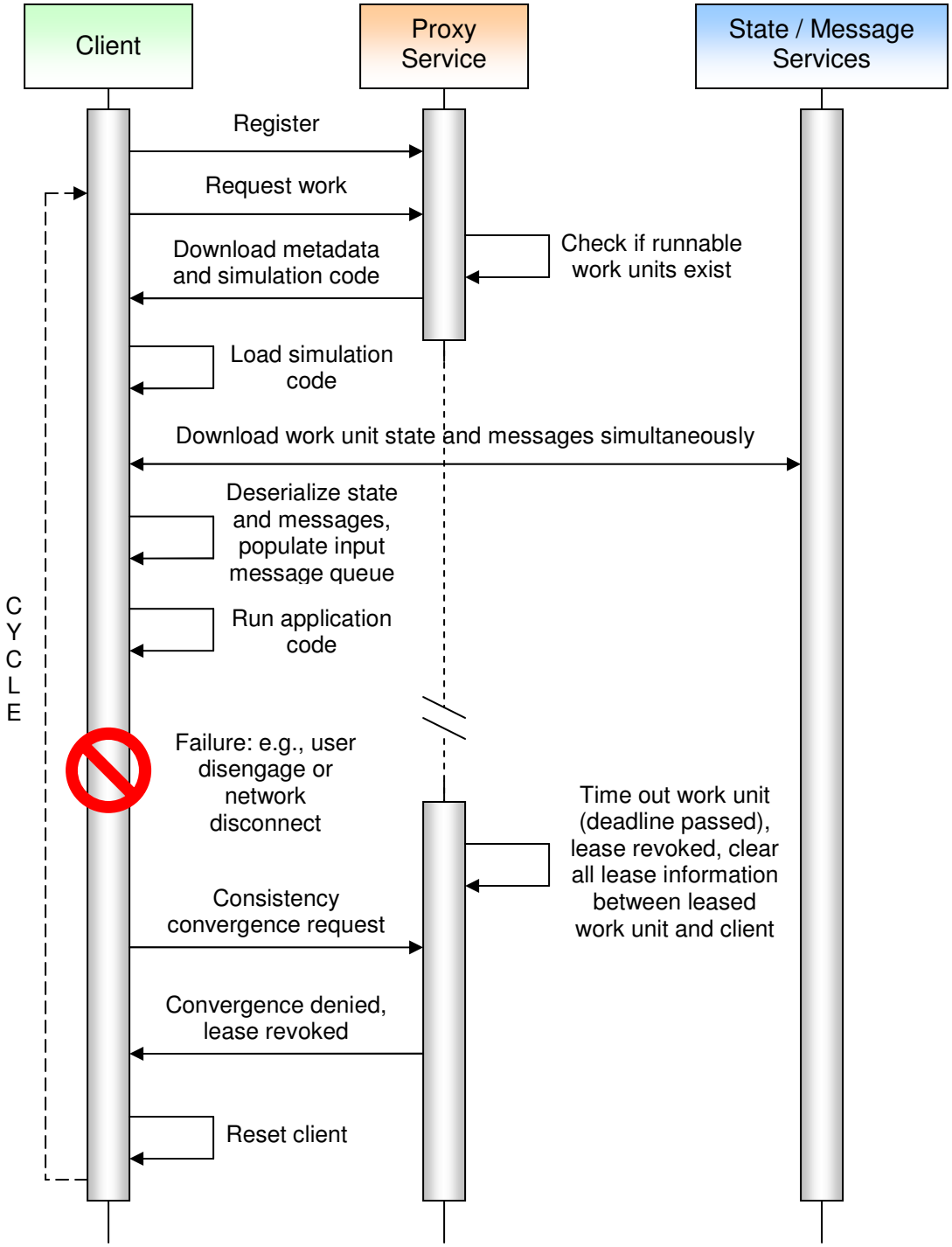

Figure 25. Interaction in the Presence of a Client Failure

The work unit request and download phase is comprised of five steps. The first step is to contact the proxy and receive a client key authorization. Client keys are unique

identifiers issued to the clients and are used by the client to identify itself in future communications. Once a unique client key is issued, the second step is for the client to perform a work unit request where the proxy may or may not lease a work unit to the requesting client. If a work unit is available to be leased, associated metadata about the work unit is downloaded from the proxy service. This third step also includes creating any necessary internal data structures to host the pending work unit. In the fourth step, the client contacts the designated work unit state server for the packed state vectors from information received in step three. Concurrently, the client contacts the appropriate message state server and downloads messages destined for LPs contained in this work unit. The fifth step sends the downloaded state vectors to the application-defined `UnpackState()` method to initialize the LPs with the proper state variables. Likewise, the messages downloaded in the previous step are unpacked and populated in the incoming TSO message queue.

After the work unit setup has completed, the application-defined simulation execution method `Run()` is invoked and the simulation runs for the entirety of the simulation execution window as specified by the request phase of the work unit setup. The final work unit phase is initiated when the simulation computation reaches the end time.

Work unit finalization and upload consists of six steps. The first step calls the application-defined `PackState()` for packaging the final state into a contiguous area of memory for upload to the work unit state server. In the second step, the client packages any remaining unprocessed messages in the incoming TSO message queue that may have been generated through self-message sends. These messages are generated

during the simulation execution from any LP within the work unit to destination LPs residing in that same work unit. In the third step, the client initiates a consistency convergence to the proxy service. Consistency convergence is a protocol that defines a set of required updates from the back-end storage services acknowledging receipt of new messages that have been generated during the client lease execution along with updated state vectors. The consistency convergence also locks down the work unit so that it will not be leased again or updated in multiple lease scenarios. This allows an atomic work unit commit, preventing inconsistent updates of work unit and message states. The fourth step involves the client initiating an outgoing message collation process where future messages destined for work units that reside on the same physical message state server are packed together. This process leverages the data gathered by background message state server lookups performed by the auxiliary thread during the simulation computation. This reduces the frequency of smaller message updates and allows the client to update groups of messages in large blocks. After this process completes, the fifth step includes uploading the final state and packed messages in step two to their respective servers. The sixth and final step is a verification of consistency convergence from the proxy that the process was completed successfully without errors on the back-end services. During steps four and five, the state and message state servers send messages to the proxy specifying updates to their respective states. The proxy acknowledges these messages and keeps track of the consistency convergence process with the final result of the update returned to the client in the final step of the finalization process.

As shown in Figure 25, client failures (e.g., temporary network disconnects or user disengages from the simulation) are detected by the proxy when it does not receive a

work unit finalization request before its wall clock deadline. These work units are reset by the proxy service and re-leased to other clients. Communications problems, such as a broken socket will cause the proxy to invalidate the lease to that particular client. User-based interrupts are handled through signal handlers and client unregistration requests, if required, are sent to the back-end system. If clients fault due to an application bug, the client must be restarted by the user, however, the proxy treats such crashes the same as ephemeral errors with work unit lease timeouts and lease revocations.

3.5 Distributed Back-End Fault Tolerance Subsystem

In contrast to the monolithic web services based master/worker PDES fault tolerance system, a variety of new issues must be addressed that pertain to a distributed back-end infrastructure present in Aurora2. The goal of this fault tolerance system is to provide transparent recovery from failure given sufficient backup resources have been allocated. In order to precisely map the range of failures covered by the fault tolerance subsystem, assumptions about the operating environment and nature of failures are presented. First, the Aurora2 system is targeted towards a trusted desktop grid infrastructure where it is assumed that workers provide results in good-faith and there is no intentional malicious behavior by the client worker pool. Thus, the fault tolerance system does not handle Byzantine failures and computational faults where clients provide intentionally incorrect results. Second, there are sufficient resources allocated for replication. Third, watchdog processes are employed. The level of sophistication used by the watchdog is relatively unimportant, with a simple shell script with process monitoring and restart capability is the minimum requirement. A minimum of one

watchdog process on the broker service is required, however, additional watchdog processes on other back-end services can provide for continual, long-term monitoring and restart capability. Finally, proper measures are taken to accommodate communication to back-end services through firewalls if necessary.

3.5.1 Broker and Proxy Interaction

The broker service, in addition to the role as a directory and locator service, acts as a controller over all proxy services. At startup, proxy services register themselves with the broker service. The broker service maintains a proxy list where it designates one proxy as the master proxy and all other proxies as slaves for replication. If the broker crashes, the watchdog service restarts the broker service. Proxy services will periodically monitor the broker host address for a live service. If the broker has no record of the proxy service, the proxy will re-register itself with the broker service. The broker acts as a multicast router for fault tolerance updates. The master proxy will periodically upload metadata tables containing client and simulation package information to the broker. The broker will then propagate this fault tolerance update to slave proxies for replication. This reduces the bandwidth requirements for fault tolerance updates on the master proxy that must continually service requests from active simulations. If the broker detects a failed master proxy via periodic heartbeats, it will promote an up-to-date slave proxy as the new master proxy service. In the case of network congestion preventing a heartbeat response and there exist two master proxies, the broker will forcefully demote a master proxy to slave status.

3.5.2 Storage Service Replication

The state and message storage services are replicated for fault tolerance. A storage service may be designated as primary or backup by the user instantiating the service. However, when a storage service registers with the proxy service, a preferred backup service may be designated as the primary service if too few primary resources exist. The primary storage service will periodically serialize simulation data and transmit them to registered backup resources when triggered by the master proxy service. The master proxy continually monitors primary state and message servers via heartbeat mechanisms. If the proxy detects that a primary resource has failed, a promotion message is sent to an up-to-date backup service. All old metadata linking the failed primary service such as unique identifiers and host addresses are replaced with the newly promoted service. Clients will automatically re-query the master proxy service for the new host address when contact with the failed primary storage service is unsuccessful.

3.5.3 Client Failures

There are no special data structures or mechanisms for client failures, as it is expected that the client pool is volatile in a master/worker system. Client leases that are not returned within the specified wall clock lease window are considered to have failed and the lease is invalidated. The computation is then subsequently leased to an available idle client worker. Clients which appeared to have failed, but are simply lagging behind due to insufficient processor speed or memory are treated the same as a failed client and the work unit return is refused, and the lease is retracted.

3.5.4 Portable Fault Tolerance

Any class instantiated by a service or class data that cannot be properly reconstructed at service creation or is critical for a service restart must implement *Serialize* and *Deserialize* routines. POCO routines are used to encapsulate data and store proper byte-ordering information. Hierarchical serialization and deserialization is employed for all back-end services. For instance, the proxy service contains various managers. During a fault tolerance update on the master proxy service, the Serialize method is called on itself, then each manager's Serialize method is invoked, which in turn may execute Serialize methods for objects that may exist within each manager. Data serialization is run using a dedicated timed thread which removes fault tolerance updates from largely interfering with a running simulation outside of common data locks. Data can be saved to a backing store such as local disk as an optional runtime configurable parameter.

3.5.5 Infrequent Checkpoint Intervals and Fossil Collection

Due to the radically different applications a master/worker PDES system must support than typical distributed task parallel applications that may only return a few (simplified) results over time, Aurora2 cannot provide fault tolerance on a per-change basis; otherwise the cost for fault tolerance would be overwhelming. If a failure occurs, the proxy may rollback the simulation to a known system-wide consistent state. However, snapshots of simulations cannot be stored indefinitely and eventually back-end services will exhaust all available memory. Similar to optimistic synchronization fossil collection mechanisms, a technique based on global control is used to reclaim memory for fault tolerance purposes.

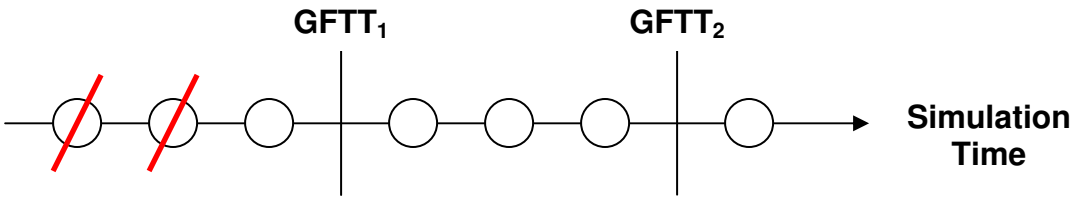

Figure 26: GFTT and Fossil Collection

The *Global Fault Tolerance Time* or GFTT is simply the GVT of the simulation at the fault tolerance snapshot time. A pair of these values determines which old fault tolerance data can be safely discarded, and is guaranteed not to cause active simulations to fail due to missing state or message data as illustrated in Figure 26. Upon successful propagation of the master proxy fault tolerance data to slave proxies, a GFTT update message is generated for all state and message services. This GFTT update will force a fault tolerance update to occur on the receiving service. Any messages with a send time less than the first GFTT can be safely reclaimed. As for states, if there exists a state entry at the GFTT time, this state must be preserved, or if a state does not exist at the first GFTT time, the first state with a timestamp less than the first GFTT must be saved. All other previous state entries can be safely discarded. A pair of GFTT values must be used to avoid GFTT inconsistencies that are a possibility that the simulation can be in an non-recoverable state if a failure occurs during a GFTT update.

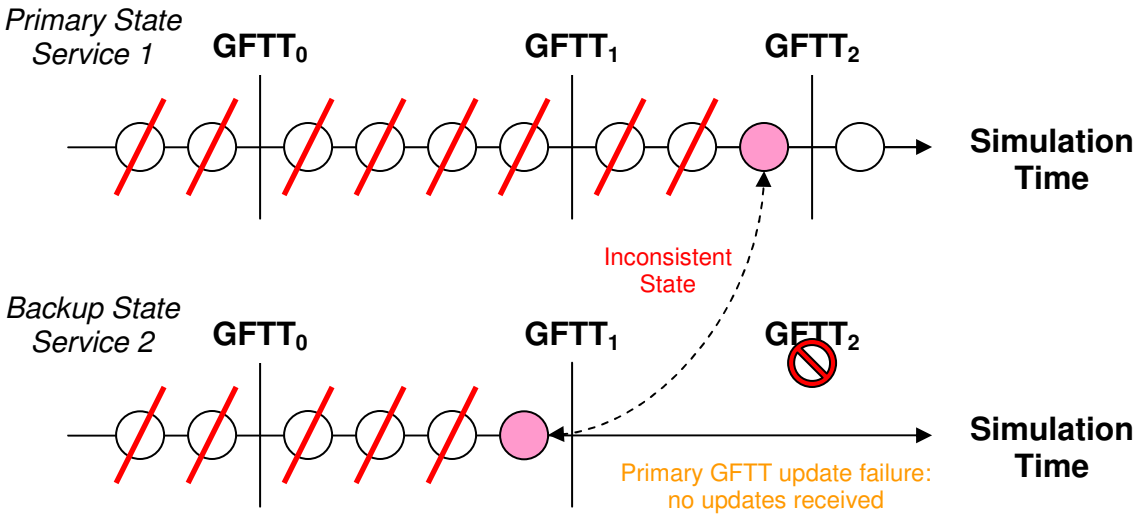

Figure 27: Non-Pairwise GFTT Updates and Inconsistencies

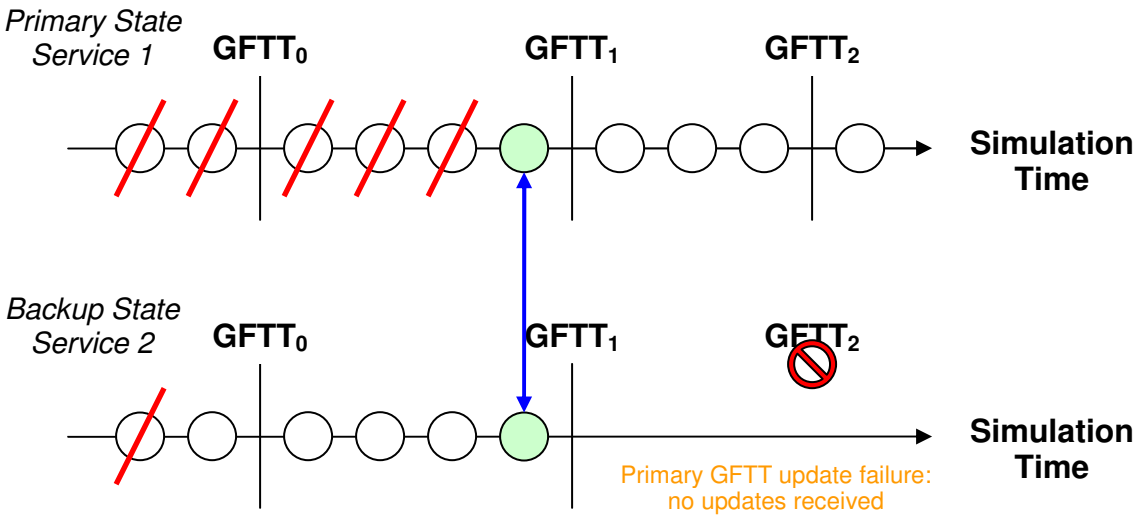

Figure 28: Recovery in the Presence of a GFTT Update Failure

Suppose only a single GFTT value was used for fault tolerance and fossil collection purposes. If a failure occurred during a GFTT update, but some services already reclaimed old state and messages, then the system could be potentially in an inconsistent state where backup services for the failed primary have not received new

state updates as shown in Figure 27. Without a forward state to jump to, nor a previous state to restore, the system could not recover from the failure and would have to be restarted.

By utilizing a pair of GFTT values, a state will always be available for restoration. This pair-wise system prevents any saved state from being reclaimed between the latest GFTT and previous GFTT values as depicted in Figure 28. Although GFTT update #2 failed to propagate to the backup state service from the primary, consistency between state service 1 and backup service 2 is maintained, allowing the master proxy to rollback the simulation to the state just prior to the GFTT update #1. This pair-wise GFTT protocol prevents inconsistencies where no available states are available for simulation restoration after a failure, resulting in a simulation restart.

3.6 Performance Study

A performance study was performed to evaluate various application characteristics for viability under a master/worker PDES system in a loosely coupled computing environment. This study provides empirical data to validate the proposed desirable properties of conservatively synchronized applications using the previously discussed performance metrics that divide processor time into relevant computation and overhead components. In addition, scalability of the system is demonstrated through the ability to deploy any number of back-end services to meet the demands of different workloads even in the presence of PDES performance-unfriendly codes.

The Aurora2 system was compiled using gcc 4.1.2 with -O2 optimization flag. Nodes designated as *Xeon* consist of dual processor Intel Xeon CPUs ranging from

2.8GHz to 3.06GHz with SMT (Hyperthreading) enabled and 1-2GB of memory. Nodes designated as *Pentium-III* consist of 8-way Pentium-III 550MHz CPUs with 4GB of memory. All machines use a GNU/Linux 2.6 series kernel and interconnected with Fast Ethernet. Although the Aurora2 system is able to utilize non-dedicated machines, for this performance study it was necessary to isolate these machines from external factors such as variable user load to obtain benchmark data without perturbations.

Compared to the previous performance study under a web services framework, the overhead is divided into more precise measurements. For the figures below, *deferred* refers to the amount of wallclock time in seconds a client spends waiting for a valid work unit lease from the master. *Import* indicates to the amount of time the client spends downloading work unit metadata, work unit state vectors, messages and the associated time spent in the application-dependent de-serialization routine. *Finalize* is the time to perform the logical inverse of *import* where the work unit state and messages are serialized and consistency convergence is achieved on the back-end services for the returning work unit. *Application* denotes the time spent executing application-dependent simulation code.

3.6.1 Synthetic Workload Analysis

To provide a broad overview of system performance, a simple synthetic benchmark was used where LPs (or work units, as this simulation maps a single LP to a single work unit) are arranged and connected in a circular fashion, where messages are sent to the immediate left or right of a work unit. This simulation contained 20 work units with uniform lookahead connectivity. This workload minimizes simultaneous activity on the back-end system by performing the simulation with only one *Xeon* client

in a sequential fashion. This exposes pure performance metrics for evaluating the interplay between absolute workload and overhead of state maintenance.

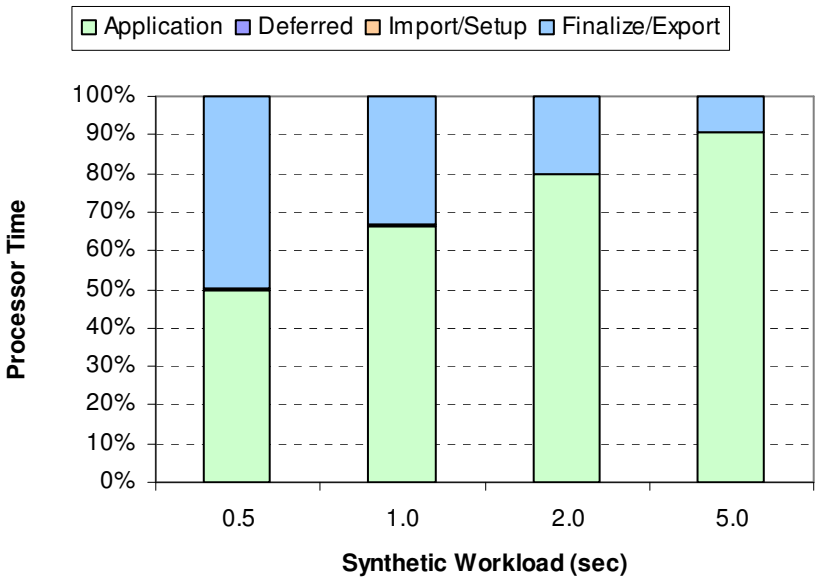

Figure 29: Effect of Workload with 10KB of State

This application mimics a packing/serialization routine associated with cross-platform master/worker systems that is determined by the amount of data transferred. Thus, as shown in Figure 29, the finalization/export phase is significantly longer than the import/setup phase. As expected, with very little state overhead, even with a low amount of computation per lease, the processor time dedicated to the application is approximately 50% or higher.

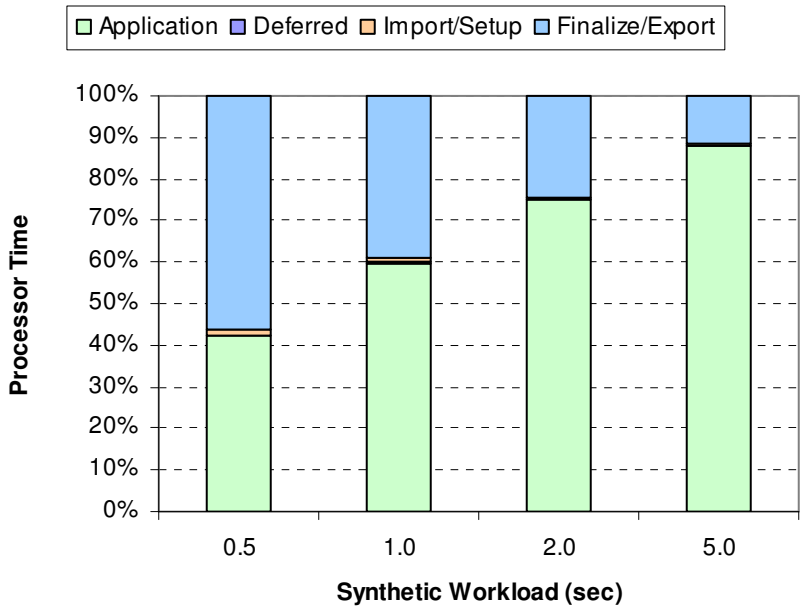

Figure 30: Effect of Workload with 1MB of State

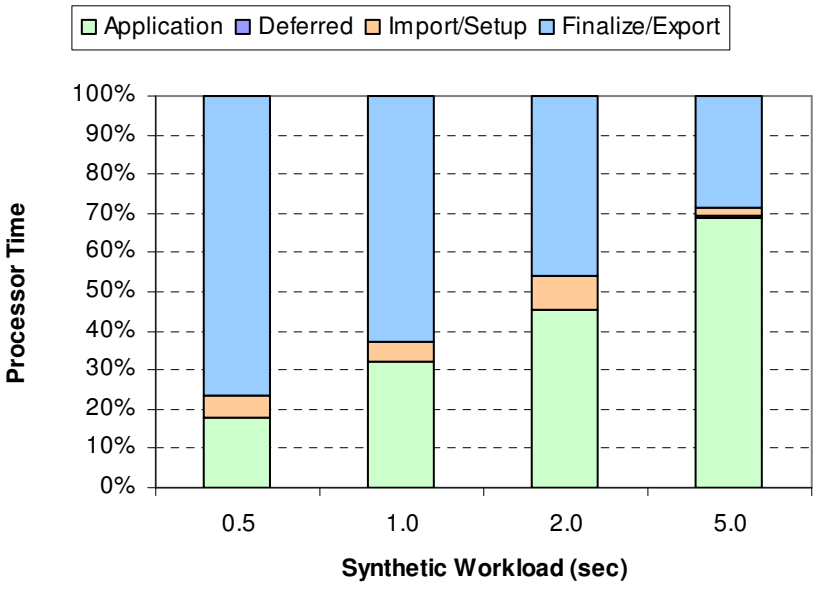

Figure 31: Effect of Workload with 10MB of State

More complex PDES codes may have larger state vectors for each lease. Figure 30 attempts to characterize applications that require larger states. As expected, the

amount of relative processor time for the application decreases as the import/setup and finalize/export phases consume more time.

Figure 31 illustrates applications with even larger amounts of state per work unit lease. The computational workloads should exceed one second per lease in order to gain significant amount of application runtime over the cost of shuffling state vector data and associated deserializing/serializing penalties.

3.6.2 Analysis of PDES Properties on Performance

To evaluate key parameters in PDES codes, a torus queuing network simulation is used. The queuing servers can be aggregated into subnets that are mapped to a single work unit. The queuing network is configured as a 25,600 server 160x160 closed torus network partitioned into 64 20x20 torus subnets that can be leased as work units. The internal links within each work unit are set at a delay of 1 millisecond. The job generator creates local jobs with random server destinations that exist within the work unit and additional remote jobs with random server destinations that are external to the work unit. Jobs reaching their destination server are assigned a new random destination according to their previous local or remote designation to keep the relative amount of local and remote jobs consistent. Job service times are exponential distributed with a varying rate parameter (in seconds), λ, depending upon the test performed. All queuing network performance tests were run on 20 clients over 10 *Xeon* nodes.

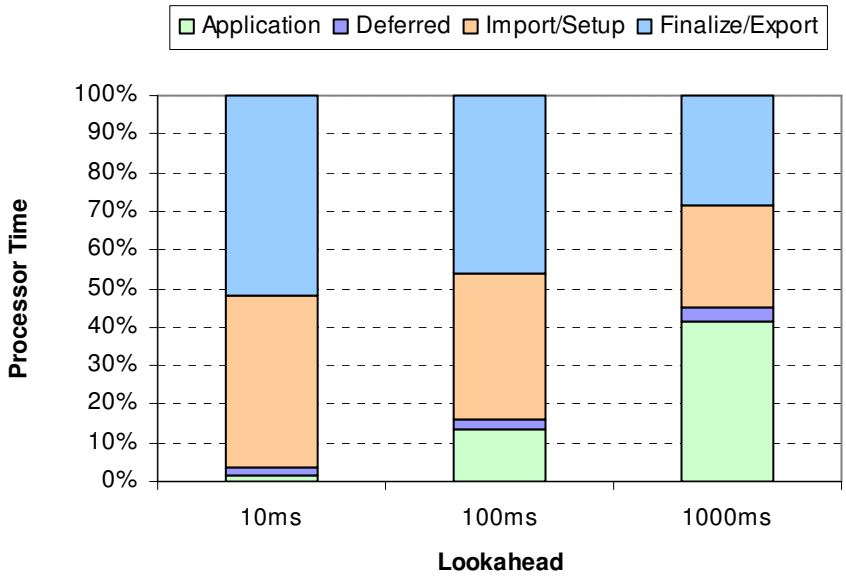

Figure 32: Effect of Lookahead ($\lambda = 1$)

To evaluate the impact of lookahead on performance, the relative ratio of 50% local and 50% remote jobs generated remained constant for each lookahead test. Similar to conservative PDES codes in general, there is a strong relationship between lookahead and the amount of processor time dedicated to the application as shown in Figure 32. The rate parameter of this simulation is 1 which leads to a high probability of event timestamps exceeding the leased time window.

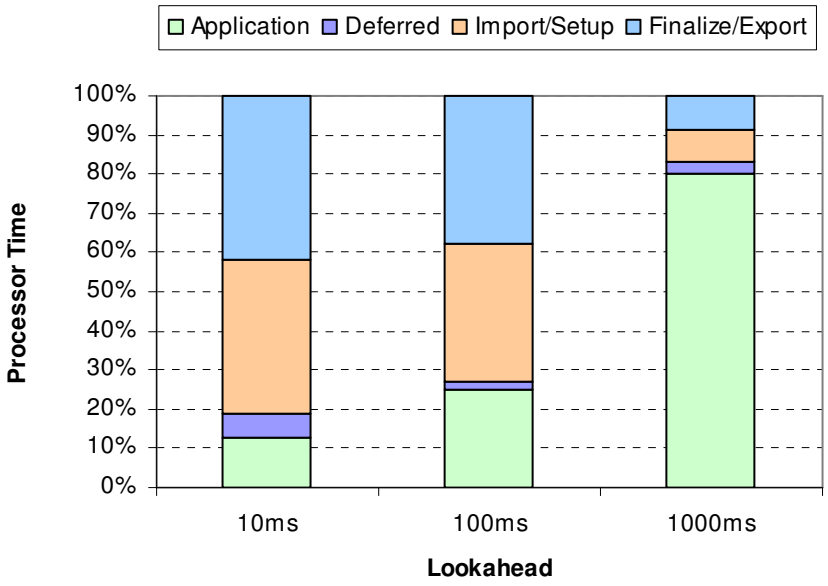

Figure 33: Effect of Lookahead ($\lambda = 10$)

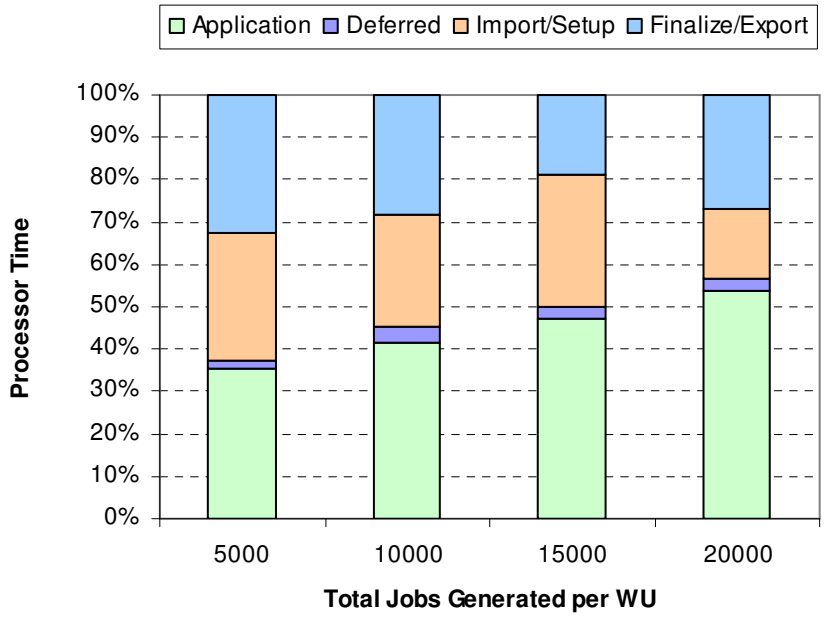

Figure 34: Effect of Absolute Workload ($\lambda = 1$, 50% local:remote job ratio, LA = 1s)

In Figure 33, the rate parameter is modified to generate timestamps with a higher probability within the leased execution window, the effect of lookahead is greatly

pronounced as more events can be processed before state and messages must be returned and synchronized with the back-end system.

Figure 34 shows the effect of total number of initial jobs generated per work unit lease, which are kept in the system due to the closed nature the queuing network. Similar to changing the rate parameter in the lookahead tests, as the number of jobs increases, the application processor time increases as the number of effective processable events within the system per work unit lease rises. However, generating too many events leads to bottlenecks in other areas such as serialization/deserialization times during setup and finalization along with increased bandwidth usage.

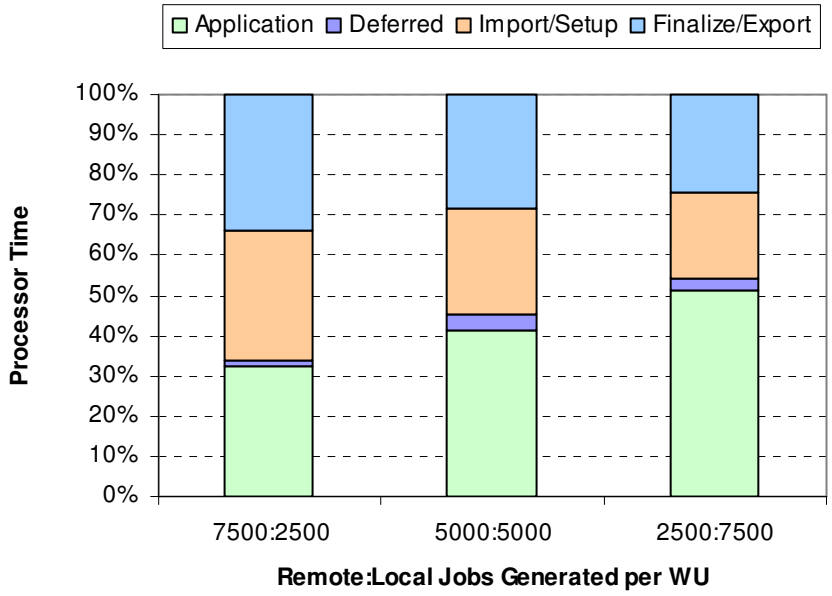

Figure 35: Effect of Relative Workload ($\lambda = 1$, LA = 1s)

In contrast to absolute workload, if the amount of work generated on the initial work unit lease remains constant but the amount of work that remains internal to the work unit is increased in comparison to remote jobs that inevitably cross work unit

boundaries and thus invoking inter-work unit message sends incur more communication costs. This diverts processor time away from application time as show in Figure 35.

The empirical data gathered from this performance study quantify our expectation that favorable lookahead, computationally intense work units, low communication costs compared to computation, and compact state and message sizes lead to desirable PDES application properties that tend to operate effectively on a master/worker PDES work distribution system.

3.6.3 Comparative Performance Study

In order to validate the scalability of this new architecture, the performance of the Aurora2 system was evaluated comparing the first generation Aurora system based on web services. Two main applications were utilized in this study that includes a torus queuing network and an existing physics simulation modeling a one-dimensional hybrid shock using the piston method. The original Aurora system was compiled with gSOAP 2.7.6c. In the figures that follow, the *XwYm* indicates *X* work unit state servers and *Y* message state servers used in Aurora2 multi-server tests.

In these torus queuing network simulations, servers can be aggregated into subnets as LPs that can then be mapped to individual work units. The first test examines the system performance of *computationally intense* work units where the number of servers within each work unit is high and the lookahead is favorable. This *computationally intense* scenario is configured as a 250,000 server 500x500 closed torus network with 625 partitions as available work units of 25x25 torus subnets. The internal links between servers within each work unit have a delay of 10 microseconds while delays between servers spanning work units have a delay of 1 millisecond. There are

20,000 initial jobs generated with 50% of the jobs destined for servers within each work unit and 50% destined for servers external to the work unit. Job service times were exponentially distributed with a mean of 5 microseconds. The following tests were performed with 42 clients consisting of 2 *Pentium-III* nodes and 13 *Xeon* nodes.

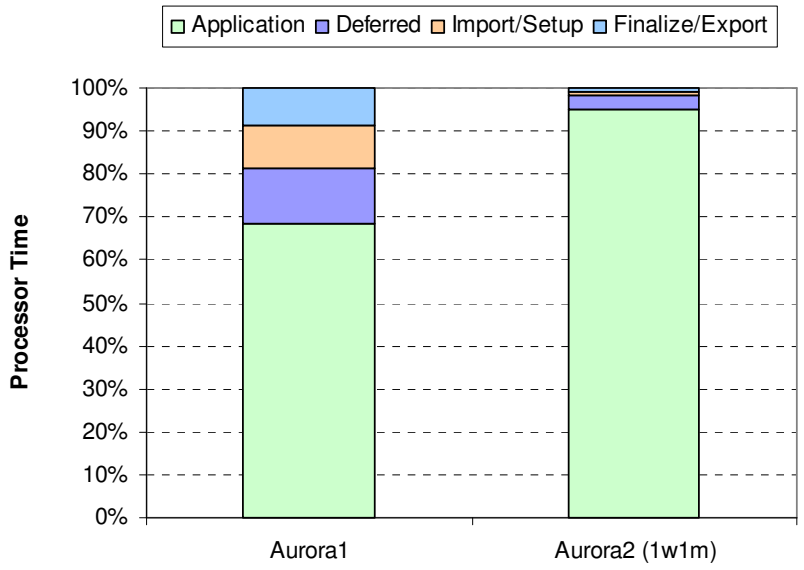

Figure 36: *Computationally Intense* Scenario Aurora Comparison

Figure 36 shows the contribution of each portion of the work unit lifecycle to overall processor time usage. The first generation Aurora system shows 68.6% of the time spent doing useful computation while 31.4% of CPU time is spent in overhead or waiting for a valid work unit lease. The Aurora2 case with one work unit server and one message state server shows approximately 94.9% of the processor time dedicated towards the application and only 5.1% for overhead. This particular simulation exhibited high concurrency and relatively large amount of time spent in simulation computation per work unit lease. Moreover, the amount of message state shifted per work unit lease

averaged approximately 60MB. Two improvements in the Aurora2 system are shown here. First, the XML encoding and processing overhead for large transfers increases the import and finalize times for the web services based Aurora system. Second, the optimized Aurora2 back-end services with multi-threading allows concurrent service requests while potentially large memory copies are being processed.

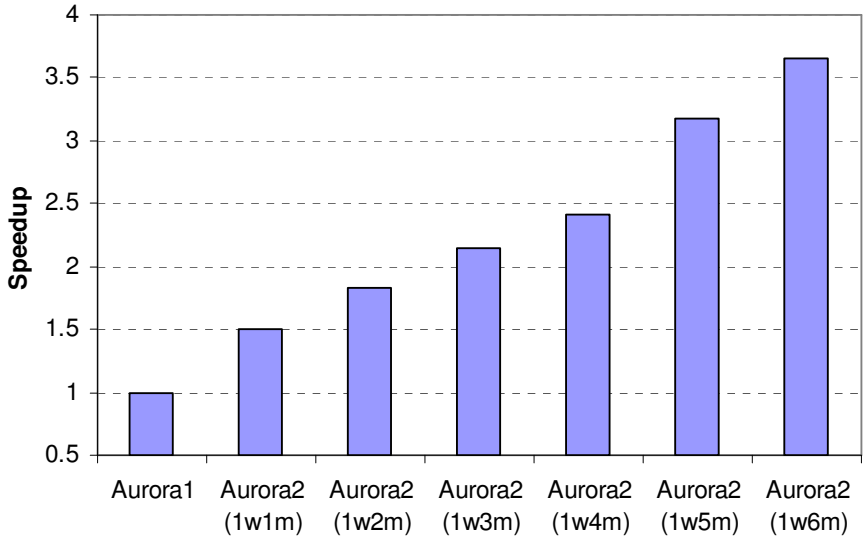

Figure 37: *Computationally Sparse* Scenario Speedup

The next test modifies the torus queuing network simulation parameters to unfavorable conditions with low concurrency even for traditional PDES systems. This *computationally sparse* scenario consists of a 150x150 closed torus queuing network containing 22,500 servers partitioned into 225 10x10 subnets which can be leased as work units. The external work unit to work unit delay has been reduced by an order of magnitude over the coarse scenario to 0.1 milliseconds. All other parameters remain the

same. This test was run over 64 clients consisting of 4 *Pentium-III* nodes and 16 *Xeon* nodes.

This *computationally sparse* scenario exhibits lower concurrency in the model due to the reduction in lookahead compared to the *computationally intense* model. There is a steady reduction in overhead as an increasing number of messaging servers are added to the back-end system. Due to the negligible amount of state vectors in this simulation, it is suitable to only add message state servers instead of a mixture of both storage services. At the sixth messaging server added, 28.1% of total processor time is recovered from overhead over the first generation Aurora system. In terms of perceived performance gain, e.g. wallclock runtime reduction, Figure 37 shows this more clearly. The Aurora2 back-end system with a single work unit server and message server is 49.6% faster than the web services based Aurora counterpart. In the final test including six message state servers a speedup of 3.65 over Aurora1 is achieved in this low concurrency scenario.

3.6.4 Scalability Analysis

To evaluate the potential for execution of large-scale simulations under a master/worker system, a particle physics simulation was chosen with characteristics conducive for traditional PDES systems. This allows a true scalability stress test of a master/worker PDES implementation taxing all available bandwidth and processing power of the back-end services. For this scalability analysis, the hybrid shock simulation is employed again. To test the effect of multi-threading in the Aurora2 system the back-end was configured with one work unit state server and message state server along with the proxy service. The threads were then varied for the test case with the following

parameters for the hybrid shock simulation: 4000 cells with 100 initial ions per cell, 0.11 lookahead, and a cell width of 0.00025. The simulation was partitioned into 200 work units. The *XpYwZm* notation represents *X* proxy threads, *Y* work unit state server threads, and *Z* message state server threads in the multi-threading tests.

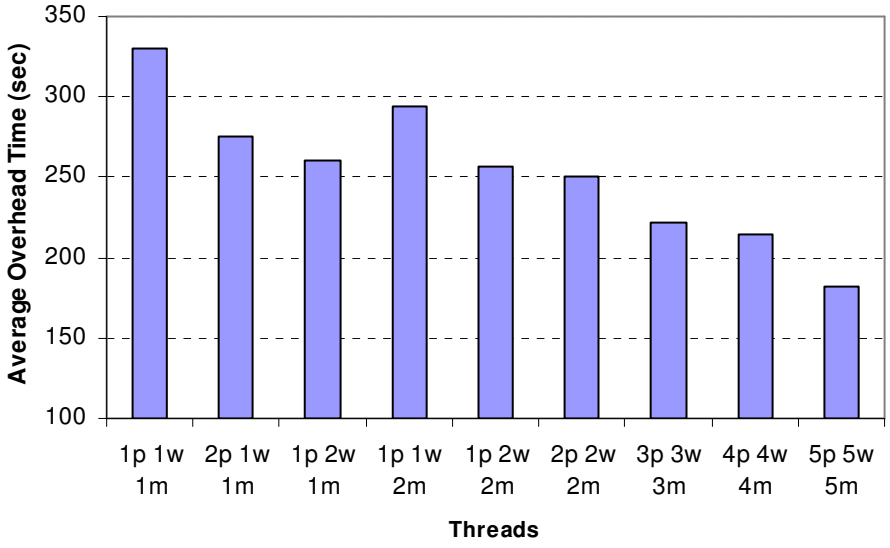

Figure 38: Effect of Multi-threading on Overhead

Figure 38 illustrates the trend of decreasing average total overhead time per client across 64 clients (4 *Pentium-III* nodes, 13 *Xeon* nodes) for this high overhead, low concurrency simulation. The ability of the Aurora2 back-end to process requests concurrently is advantageous in reducing average overhead time per client particularly in the areas of work unit import and work unit finalization.

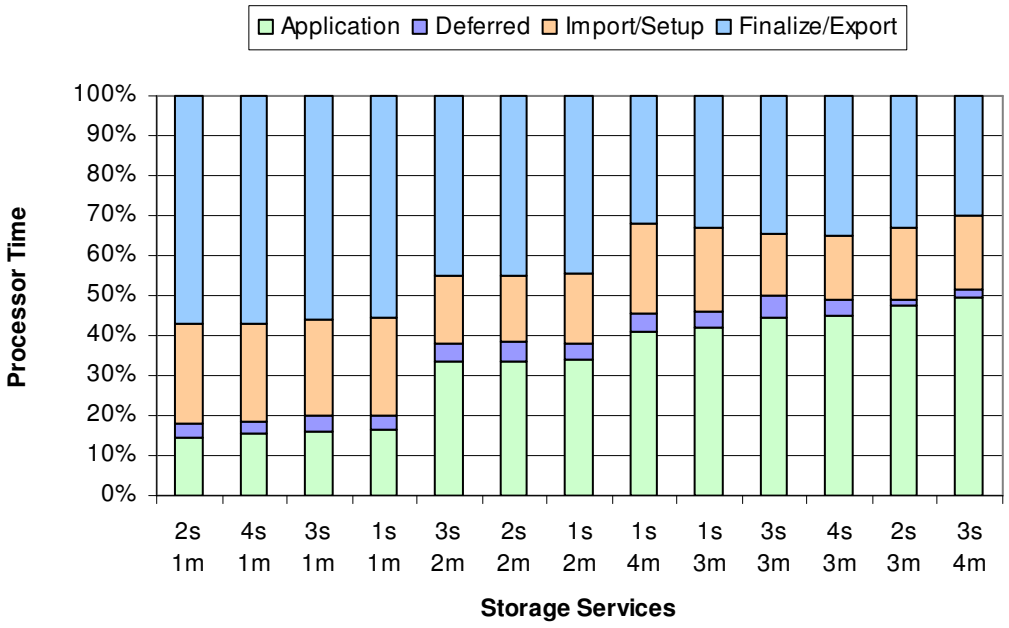

Figure 39: Hybrid Shock Multi-server Performance

The final test case involves a large-scale Hybrid Shock model that contains both large amounts of state vector and large amounts of messages being shifted each work unit cycle. The configuration for this Hybrid Shock model increase the number of cells to 10,000 and 800 initial ions per cell. The other parameters remain the same. This simulation was run over 64 clients on 32 *Xeon* nodes.

Steady performance improvement is shown in Figure 39. Although adding just message state servers produces great initial gains, the performance levels off after the third message state server if the work unit state server is restricted to only one instance. The addition of similar numbers of work unit and message state servers allows for the best scaling improved performance. Application CPU time improves from 14.6% to 49.3% in the *3w4m* case.

Performance is slightly degraded when adding more work unit servers initially over message state servers. This is due to the phenomena that as the work unit state server request latency declines, the request rate into the message state server increases reducing the responsiveness of that service further. This implies that the initial bottleneck is not the state server but the message state service for this particular simulation application, and further proven that the addition of one message state server improves performance more than two, three, or four work unit servers.

3.7 Task Parallel Simulation Support

The Aurora2 system was enhanced to allow task parallel simulations to run simultaneously with PDES simulations providing an insulated and integrated environment for mixed-mode parallel simulations across public resource computing infrastructures and desktop grids. Aurora2 allows serial simulators (e.g., executables) to be run in a parallel fashion utilizing the worker pool to perform many replications at once. Aurora2 also gives fine-grained control over simulations to allow complex operations that can break apart trials and distribute them across clients. The following is a case study on performance for an actively deployed simulation suite.

The Georgia Tech Research Institute Ballistic Missile Defense (BMD) Benchmark is a PC-based Monte-Carlo simulation tool for the development and performance benchmarking of missile-defense multi-target multi-sensor algorithms (e.g., track fusion, data association, general C2BMC techniques). Uses for this tool include development and assessment of C2BMC sensor network algorithms and algorithm specification for code development. The BMD Benchmark also provides for real-time

testing, pre-mission test event planning, real-time metrics, and post-mission data analysis. The BMD Benchmark software is a single-threaded discrete event simulation consisting of approximately 1800 MATLAB files.

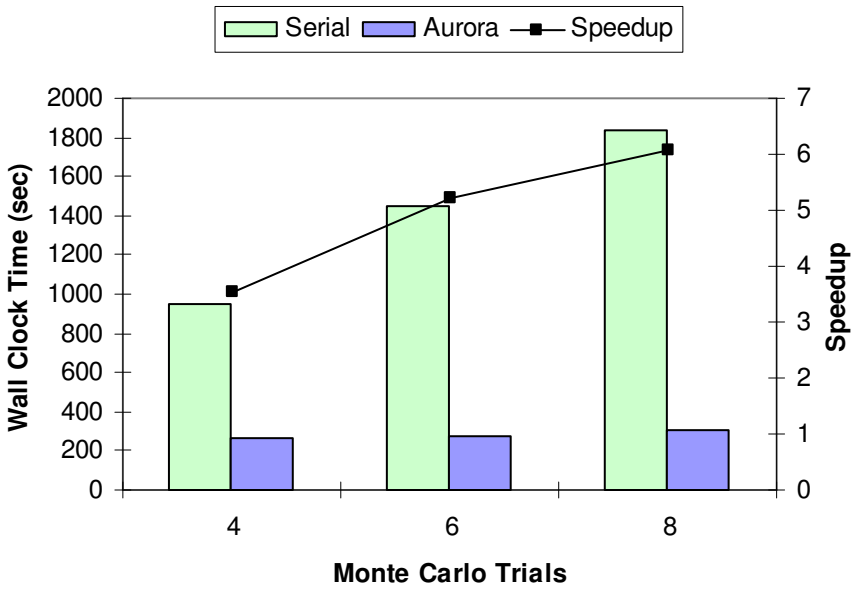

Figure 40: Task Parallel Replication Speedup

The Aurora2 system digests the input file and if set as a distributed run, will create equivalent work units to the number of monte carlo trials plus an additional work unit for an output combining phase. Using the sample scenario files provided with the BMD Benchmark, a speedup test was performed varying the number of Monte Carlo Trials. For the Aurora-based distributed run, the number of Monte Carlo Trials corresponds to the number of clients used to execute the simulation as each trial was distributed to one worker. As shown in Figure 40, respectable speedup is achieved by distributing the Monte Carlo Trials to available clients with a 6 fold speedup on 8 workers.

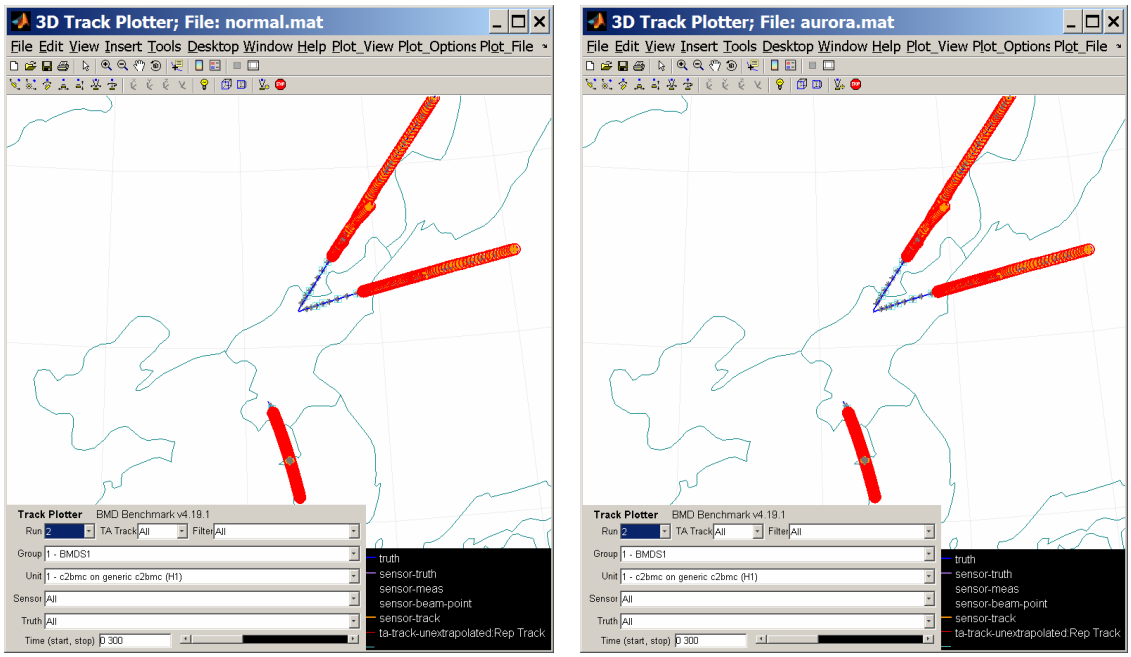

Figure 41: Validation of Results

Resultant data between a sequential run and an Aurora-distributed run are shown in Figure 41 that shows identical plotter data. Other metrics data such as position error and covariance produced identical results validating that the Aurora-distributed run produces the exact same data as a serial run.

3.8 Conclusions

The Aurora2 architecture has made significant strides in improving performance for large-scale PDES applications on a master/worker paradigm. We demonstrated that the distributed back-end system with a multi-threaded proxy, work unit state, and message state services can reduce overhead time and improve request latency. The ability to instance multiple storage services across many machines enhances the scalability of the Aurora2 back-end system. The improvements to the client by

separating auxiliary Aurora-specific tasks and the computation thread reduce overhead times.

The first generation Aurora system provided a unique approach to PDES execution through a web services communication middleware. However, this prototype system suffered shortcomings in providing services for large-scale PDES programs. The Aurora2 system offers a more robust application independent framework that builds upon the principles of the first generation Aurora system delivering scalable higher performance. This improved architecture allows large-scale PDES applications to take advantage of the master/worker style of parallel workload distribution and execution across desktop grids and public resource computing infrastructures.

Moreover, enhancements were made to the Aurora2 system to incorporate an insulated and integrated environment for task parallel simulations. This interface allows even fine grained control to allow distribution of replications to the worker pool in a mixed-mode execution environment simultaneously supporting both PDES and task parallel executions while providing features such as automatic cleanup phases and final output file recombination.

Although significant improvements to the master/worker PDES architecture were implemented, there are still areas where improvements are needed. The overhead-prone areas of state vector transfer and message updating can be further optimized to reduce the performance gap of master/worker PDES to traditional PDES systems. Moreover, work units can be scheduled in a more optimal fashion to reduce the total amount of leases required to complete the simulation as well as reduce deferred wait times. Optimization

approaches and associated issues they intend to solve and create themselves are explored in the next chapter.

CHAPTER 4

REDUCING INTRINSIC OVERHEADS IN CONSERVATIVELY SYNCHRONIZED MASTER/WORKER PARALLEL DISCRETE EVENT SIMULATION

Today, applications for public resource computing and desktop grid infrastructures are largely limited to embarrassingly parallel codes. These codes inherently scale well over many machines due to the lack of interprocess communications and synchronization. With PDES, synchrony between partitions of work is necessary for correct execution. This requirement creates a plethora of issues, where the major impact is the reduction of parallel performance due to system overheads such as transmission of state vectors and indirect message sending.

Although a master/worker approach under loosely coupled distributed computing infrastructures such as desktop grids provides potential benefits such as dynamic simulation execution, fault tolerance, and semi-automated load balancing, these advantages do not come without a cost. In exchange for these benefits, a master/worker system must pay with performance overhead where every work unit lease incurs overhead from lease setup to state vector and message download along with increased storage costs and work unit finalization overhead where the work unit state must be re-packaged and sent for storage on a non-volatile destination.

In order to reduce these intrinsic overheads, the problem is attacked using a simple process. How can a master/worker PDES system mimic a traditional monolithic

system? Two areas can be readily identified as overhead prone: state transmission and blocking time on both the lease and finalization phase along with messaging overhead. The solution approach to these issues is to amortize the cost of state vector transmission and message passing overhead. Moreover, scheduling work units in a non-first-come first-serve basis but rather identifying which work units may lie on the critical path of the entire simulation may provide overall better performance. The focus of the work presented in this chapter is to reduce the overheads inherent in a master/worker system by examining possible solution approaches and associated trade-offs and issues with each.

As previously mentioned, in order to close the performance gap between master/worker and traditional PDES our approach entails mimicking traditional protocols. Conventional PDES systems are often fully connected where all nodes can pass messages between each other. Moreover, under conservative synchronization there is no need to save state, assuming no fault tolerance mechanisms are provided. It is clear that a master/worker system will not be able to match the performance of a traditional monolithic system; however, the difference in performance can be reduced with expedited message delivery and reduced communication for state vector transmission.

We propose to reduce the intrinsic overheads under the master/worker PDES paradigm utilizing four techniques:

1. Work unit caching
2. Pipelined state updating
3. Pro-active message sending
4. Scheduling policies exploiting PDES properties

These techniques allow a master/worker PDES system to behave more like a traditional system, although not identically due to the volatility of the worker pool and the inability to directly send messages between clients (communications occurs via the master). Within each of these approaches lies trade-offs that must be carefully examined. For example, a blind caching technique where cache hits are maximized for work unit leasing can lead to increased deferred wait times if the system attempts to block for a cache hit.

Work unit caching provides a reduction in the amount of state transferred between the state storage services and clients which have valid cached data. This can prove to be beneficial if the system can utilize the cache available at the client sites and maintain a relatively high cache hit ratio. Issues that must be addressed are maintaining cache coherence and consistency as work units can migrate between clients alongside a replacement and eviction policy.

In conjunction with the caching mechanism, an appropriate state updating mechanism must be in place to allow cached copies of state on the client to be reconciled with the back-end storage services. Without a state update protocol to the backing store, a client failure can lead to an irrecoverable state forcing the simulation to restart. An efficient state updating mechanism is also needed to allow the work units to be migrated between clients through the back-end system.

Although the reduction of state vector transmission is a major component to overcoming the overhead barrier, simulations that exhibit high interprocess messaging relative to state vector size will not exhibit significant performance gains through a caching mechanism alone. A pro-active approach to message sending where messages

are collated and sent in groups before the work unit reaches the end time of the lease window allows overlapping communication with computation.

Finally, enhanced scheduling policies that exploit PDES properties such as lookahead and time window information can prove beneficial for increasing concurrency in the system. Arbitrary work unit leasing can lead to longer simulation runtimes through shorter and more frequent leasing or increased deferred waiting times due to blocking.

References to the Aurora system refer to the second version of Aurora or the Aurora2 implementation as discussed in chapter 3 from this point forward.

4.1 Work Unit Caching

When considering a caching mechanism for a master/worker PDES system, a variety of issues must be addressed. While there are related issues shared among distributed shared memory and proxy caching systems, special attention must be paid to issues specific to master/worker PDES systems. There are four principles in caching systems where issues must be addressed:

1. Locality
2. Coherence and Consistency
3. Eviction and Replacement Policy
4. Network Policy

Differentiating concepts that apply to master/worker PDES systems when coupled with these traditional caching principles will be discussed highlighting the importance of new mechanisms which exploit PDES-specific properties such as connectivity, predictability, and time stamps.

The granularity of a cache block is considered the packed state vectors of the work unit. This data is cached on the worker with a work unit identifier and timestamp tags. Because the space for cached data is limited, a replacement policy and eviction mechanism is required. In a master/worker PDES system, only a single writer is assumed where there may be a single reader (SRSW) or multiple readers (MRSW).

4.1.1 Locality

Locality is usually separated into two different categories under traditional caching mechanisms: spatial and temporal locality. References that may be nearby to a recently referenced datum or instruction is referred to as spatial locality while the probability of referencing the same datum or instruction at some point in the future is referred to as temporal locality. In PDES, both locality types can be exploited. Locality of reference for PDES codes can be separated into two different types for both temporal and spatial variants:

1. State vector locality
2. Output message locality

Temporal locality is an important principle in any caching system and is fully exploitable in master/worker PDES systems. Since workers (i.e., clients) execute available work units as directed by the master service they may perform computation over a wide range of work units available in the system. An obvious optimization exploiting temporal locality is to store finalized state vectors (e.g., the state upon reaching the end of the leased execution window) locally at the client in addition to updating the state on the back-end service. Future leases can be directed to clients that have valid state vector caches. This will reduce overall transmission overhead and conserve bandwidth.

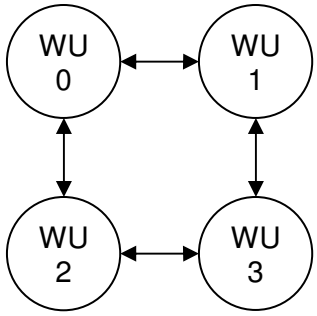

Figure 42: Sample Four Work Unit Connectivity

In the following discussion a simple four work unit connectivity graph as shown in Figure 42 will be used for illustrative purposes. Here we assume that work units communicate to each other through time stamped messages and are only processed in increasing timestamp order when it is safe to do so, preserving the local causality constraint. As depicted, message paths are directed and work units may only send messages directly to those to which they are connected. Therefore, work unit 0 cannot directly communicate with work unit 3, nor can work unit 1 communicate directly with work unit 2 and vice versa.

In contrast to state vector caching, a specialized form of temporal caching utilizing message semantics can be deployed for master/worker PDES systems. For codes that generate many messages within a possible future execution time window it may be beneficial to cache such messages locally. For example, if work unit 0 sends 5000 messages to work unit 1 for which 4000 messages fall within work unit 1's current simulation time plus the input lookahead from work unit 0, then it may be beneficial to cache these messages for a directed "future cache hit" of a work unit 1 lease to the current client which completed execution of work unit 0.

Although temporal locality is important for any caching system, PDES codes can exploit spatial locality in a much more predictable fashion than traditional systems such as CPU or web caching systems where there only exists a probability of a cache hit. In a master/worker PDES system, a cache hit exploiting spatial locality can be directed and guaranteed as application-specific information such as connectivity graphs are exposed to the underlying master services.

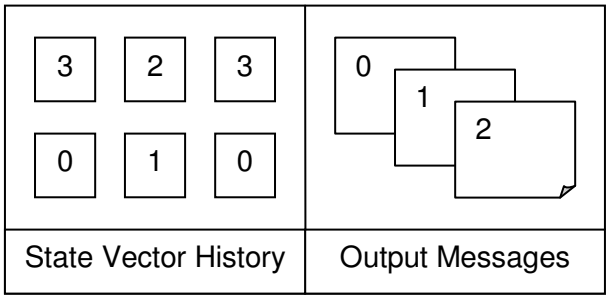

Figure 43: Client Internal State

Figure 43 depicts a sample internal state of a worker client, where a state vector history is stored in most recently used order (from left to right) and the current output messages. As shown in the figure, the last work unit processed was work unit 0 with messages generated for work unit 0, 1 and 2.

For the following case, assume that work unit 0 is currently being processed by the client and that the previous execution of work unit 1 has left a valid cached state vector data. Since it is known that work unit 1 is directly connected to work unit 0, the system can exploit this fact and cache output messages to work unit 1 that are generated by work unit 0. By employing a reservation system, the master services can generate "extended" cache hits which not only include state vector data but also message data. This reduces the number of messages that must be downloaded by the client.

For the next case, suppose that state vectors for a particular PDES code are large, thus incurring significant transmission time. Instead of simply assigning leases according to whatever work unit is available, the aforementioned reservation system coupled with a pre-fetch mechanism can be employed. Assume work unit 0 and work unit 1 are concurrently leased and work unit 2 is blocked in an idle state and cannot proceed until work unit 0 completes and returns data back to the master services. In this case, the master can exploit spatial locality by reserving work unit 2's future lease for the client currently executing work unit 0. While the client is executing work unit 0, work unit 2's state vector data is pre-fetched and updated from the back-end service. Since connectivity between work unit 0 and 2 is known a priori, any output messages generated by work unit 0 to work unit 2 can be cached alongside the pre-fetched and updated work unit 2 state. Upon completion of work unit 0, all pertinent state and some of the messages are already local to the client providing a large cost savings in overhead and bandwidth.

Temporal or spatial locality may vary widely between simulation models and special consideration must be given to favoring a certain locality of reference over another. In a PDES system, a priori knowledge of connectivity and predictability within the model can be fully exploited to maintain a high cache hit ratio.

4.1.2 Coherence and Consistency

Cache coherence and consistency in typical caching systems such as distributed shared memory are varied ranging from snoopy to directory-based. In a master/worker PDES system, a directory-based coherence system is the most similar, however, employing a traditional directory-based coherence protocol will not work under a

master/worker PDES system. Since two-way connections between workers and the master services cannot be assumed due to firewall issues (discussed in chapters 2 and 3), control messages cannot be reliably forwarded from the master to workers. In a "worker pull-based" system all communication is initiated from the clients only. Thus a hybrid directory-update coherence mechanism with a timestamp-based consistency protocol is proposed.

In master/worker PDES systems, work unit leases are directed by the master service. Although workers request work to be completed, they cannot request a specific work unit. Thus all requests are initiated on the master, where all caching information is stored with global knowledge of the entire system. Consequently there is a trade-off where cache hits can be directed to a certain extent, but state updates must always be written-through to the "backing store" (e.g., back-end state service). In a conventional directory-based coherence scheme, requests may be forwarded from home nodes to caches that contain valid data. In master/worker systems, this is not possible due to the strict restriction that clients may not communicate with each other.

Since true bi-directional communication (e.g., master-initiated as well as worker-initiated) cannot be assumed, coherence in master/worker PDES systems can only be maintained by piggybacking updates on control messages as dictated by established protocols for work unit request, lease, and updates. New cached states or evicted states with timestamp data can be transmitted to the master service to update the centrally held cache directory. Similarly, cache invalidations, update, and pre-fetch commands can be piggybacked on control message acknowledgements from the master service to the clients.

In a master/worker PDES system, leases always involve a read and a write, where the state vector data is read and fetched from the backing store and updated state is written upon completion of the lease. Thus, a singular read or write cannot happen for a work unit state and instead it is more accurate to consider states to be checked-out, modified, and checked-in by workers. Consequently, when state is updated and checked-in by workers upon completion, the master service can simply invalidate entries in the directory for which other workers hold cached data. When affected workers contact the master service, invalidated cache entry data can be piggybacked on messages back to the workers.

In a PDES system, global limits can be computed such as LBTS and GVT. Conventional uses for these values are to provide incoming message guarantees for safe execution and fossil collection. The meaning of these values can be extended to the cache consistency mechanism to include automatically invalidating cached output messages since these messages are not associated with a granular cached state.

4.1.3 Eviction and Replacement Policy

Some discrete event simulation models are large and the reason to parallelize such models is the ability to distribute the model on many processors due to memory constraints. With a caching system, available memory is a concern. Cached entries cannot persist on the worker forever and a method for selecting which entries to evict and replace can have a tremendous impact on cache hit ratio.

Traditional caching systems such as those employed in distributed shared memory utilize Least Recently Used (LRU) or variations of that policy. For master/worker PDES systems, LRU is not the best approach. As simulation time progresses away from the

stored cached state, there is a higher probability that the cached entry will be used in the future. This is due to the mechanism that governs leases to workers.

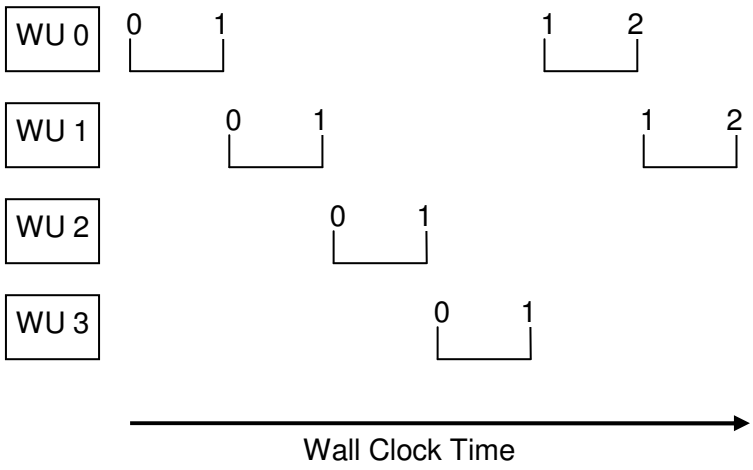

Figure 44: Example Lease Scenario for One Worker

As a sample scenario, consider the same work unit topology as shown in Figure 42 under a master/worker system with a single worker for the sake of simplicity. The lookahead between each work unit is 1 time unit and thus execution windows are 1 time unit in length as shown in Figure 44. The master progressively leases each work unit to the client as depicted. If the client is has limited memory resources and must evict a cache block, under an LRU scheme, work unit 0 would be the chosen block for eviction and replacement. However, this would result in a cache miss on the subsequent lease after the worker completes computation on work unit 3. A Most Recently Used (MRU) scheme would instead evict work unit 2 cached state and would leave work unit 0, resulting in a cache hit.

However, simple schemes such as MRU do not fully exploit the information available to the system. The master service holds a list of work units that are available

for immediate lease. This list can be used to determine which cache blocks should be preserved if possible. Work units that do not exist on this list can be evicted with minimal impact on the cache hit ratio. The predictable nature of discrete event simulations with lookahead permits this specialized form of eviction and replacement policy.

In addition to random, LRU, and MRU strategies that are well-known eviction and replacement policies, two additional policies are proposed that take advantage of simulation specific information. Earliest simulation time first (ESF) and earliest simulation time last (ESL) are two policies that prioritize cache block evictions based on the time windows. The end simulation time of the window is the timestamp that is used to define the ordering of the cache blocks. The premise of these two policies is based on the probability of a work unit becoming advanceable (or runnable) in the near future. Advanceable (or runnable) is defined as a work unit that is available to be leased under a conservative synchronization system where the next lease window end time is greater than the begin time (or current simulation time). An ESF policy would evict work units with the smallest simulation time while an ESL policy would evict work units with the largest simulation time. Thus, the ESF policy operates on the principle that work units with the earliest simulation time are most likely out-of-date and have already been leased to another client for processing. The ESL policy assumes the opposite, where work units with earlier simulation times have a higher probability of becoming advanceable in the near future and attempts to preserve those cache blocks.

4.1.4 Network Policy

In public-resource computing systems, bandwidth is a concern as workers may not have high capacity links to the master services. In a caching system, attention must be paid to properly allocate available bandwidth. Latency between request to application computation and latency from computation completion to work unit return are also concerns for minimizing overhead and increasing parallel efficiency.

When cached states are invalidated and must be updated, additional bandwidth must be allocated. Since schemes that push data from the backing store to the workers are forbidden in a master/worker system other methods must be used. One alternative is to allow clients to periodically poll the storage service once an invalidation message arrives from the master service. If spare bandwidth is available to the storage service, updated state can be piggybacked on the acknowledgment message to the worker. Pre-fetching mechanisms using a polling service may be further complicated. A cache miss can occur if not enough bandwidth is available after a pre-fetch command from the master service arrives and an update is not processed before a new lease is issued.

With global knowledge of work unit leases to workers, bandwidth can be prioritized to minimize latency. For instance, if the system is waiting on a few work units to return for forward progress in simulation time, it may be more beneficial to prioritize the work unit completion and return to the master services than allow cache updates to occur if no spare bandwidth capacity is available. A cache miss may be more favorable than that of delaying the entire system due to delayed work unit returns.

4.2 State Updates

4.2.1 Pipelined Updates

In a typical master/worker PDES system, state is packaged at the end of each work unit lease cycle and an update is sent to the master back-end service for consistency. Synchronous state updates increase the amount of overhead per lease cycle as the client must perform this operation outside of simulation computation. The basic premise on optimizing this operation is to allow state updates to be pipelined in conjunction with state caching. Instead of transmitting the state update synchronously, the state is updated in asynchronously during the next deferred wait cycle or during the next simulation application computation.

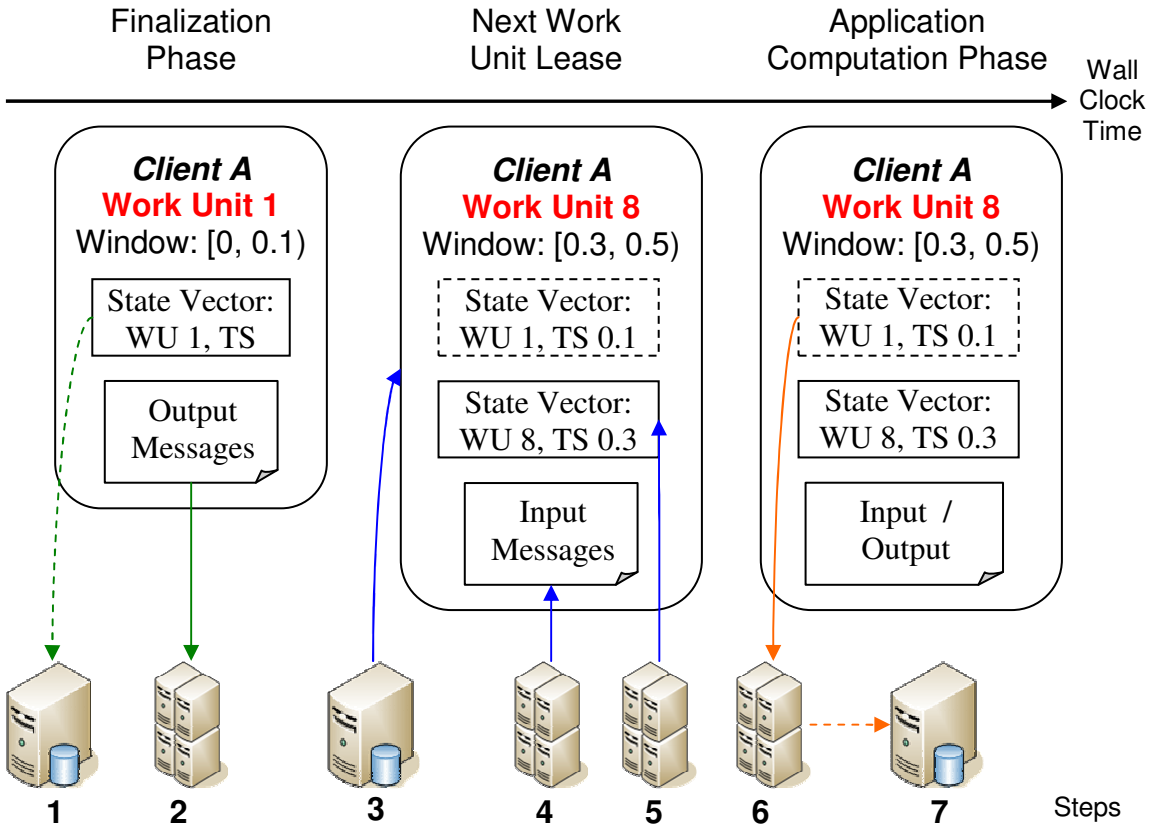

Figure 45: Pipelined State Update

The pipelined state update process is shown in Figure 45. Once client A completes processing the lease window of [0, 0.1) it begins the finalization phase. After the client is authorized to begin the update process, client A sends a lightweight update message to the proxy informing it that a pipelined state update will be performed in step 1. Output messages are uploaded normally in step 2. For the next work unit lease cycle, client A is leased work unit 8 with a window of [0.3, 0.5) in step 3. Steps 4 and 5 involve downloading input messages and the proper state vectors for the time window, respectively. During the application computation phase, client A performs a state update for the previous lease of work unit 1 updating the state to the appropriate back-end state server in step 6. The state service then sends a control message to the proxy service indicating that the state update was successful and consistency is reached in step 7.

If significant amount of application computation time (e.g., computationally intense work units) exists, then a pipelined state updating system offers tremendous advantages by masking the state update overhead over processing the simulation code. Applications that do not exhibit computationally intense work units that can provide adequate overlapping time will not exhibit performance increases through a pipelined state updating mechanism.

4.2.2 State Pre-Fetching

For simulations that exhibit favorable computation to communication ratios, state pre-fetching is an option to further enhance performance by populating the local cache with up-to-date blocks. In addition to pipelining state updates, if spare time is available during the application computation phase additional state vectors can be downloaded.

During the work unit lease, a set of pre-fetch candidates are piggybacked on the lease control message. Work units that are not advanceable (or non-runnable) are selected as pre-fetch targets. Non-runnable work units are guaranteed not to be immediately leased so there is a greater probability that the pre-fetched work unit state will not be wasted due to the work unit being leased in the near future while the current lease is being processed.

4.3 Message Updates

Under conventional master/worker systems, especially those employed in a wide-area infrastructure, data transmission usually only occurs during the work unit lease and return. Under local-area infrastructures such as desktop grids, network bandwidth may be higher with more reliable connectivity. Fully buffering messages may not be necessary under these settings and a more aggressive approach to message sending may provide performance improvements.

Three different message updating schemes were defined under the Aurora master/worker implementation: static, aggressive, adaptive. All policies implement a message buffer that temporarily stores generated messages. A static policy will push messages to the back-end message storage service when the message buffer limit is reached. The current Aurora implementation uses a 16KB static buffer limit; once the accumulated message size reaches this limit, all messages in the buffer are collated by message service and sent in bulk to the proper server. Any remaining messages at the end of the application computation phase that remain in the buffer are sent during the finalization phase. An aggressive policy attempts to send messages as soon as they are generated. In the case of simulations that send a large number of messages, a message

update may be in progress and instead of immediately sending, the message may be buffered and sent with the next message generated. The adaptive policy utilizes run-time statistics and attempts to provide an adaptive limit on when to push messages to the message service. The policy gauges the number of messages generated along with the application computation wall clock times observed in the past. If the current message generation rate matches or exceeds the past observed average message output, an attempt is made to send messages stored in the current buffer. The premise of this policy is to adaptively tune the output rate in hopes that the messages pro-actively sent are not sent too aggressively but are sent in intervals sufficiently frequently so that most of the message sends are masked by the application computation phase.

$$\varphi = \frac{nm}{\alpha L} \tag{4.1}$$

The overall average message generation rate per lease cycle (φ) is calculated through equation (4.1). n denotes the total number of messages generated in all previous leases seen by the client, m is the average message size, α is the total application computation time in all previous leases, and L is the number of previous lease cycles. Thus φ provides a message generation throughput rate, or more precisely the number of message bytes generated per second of application wall clock time per lease.

$$\upsilon_i = \frac{n_i m_i}{\alpha_i} \tag{4.2}$$

The current work unit lease message generation rate is captured by υ in equation (4.2) where i denotes the current lease cycle number. If $\upsilon_i >= \varphi$ for the current lease i, the current message generation rate matches or exceeds the average message generation rates and messages are pushed to the back-end message storage service. Additional message

buffer size checks using the metric total message size transferred per lease can force a flush of messages if the messaging rate falls below φ for prolonged periods. This prevents the message buffer from growing too large without triggering the messaging limit. If no previous runtime statistics are available, this policy falls back to the static policy until sufficient statistical data (e.g., more than 1 lease cycle with non-zero average statistical data such as runtime, and the number and amount of messages generated) has been gathered.

4.4 Analytical Overhead and Performance Models

Detailed performance models are necessary to formulate scheduling policies that are compatible with a master/worker PDES caching mechanism. The models presented here expand on those presented in chapter 2, with more detail for each overhead component.

4.4.1 State Transfer

State vector transmission time is an important component of overhead. This cost is incurred twice in a master/worker system per lease cycle. The first time is when the state is downloaded from the back-end storage service to the client. The state is unpacked at the client invoking application-defined deserialization routines. When the client reaches the end time of the simulation lease window, the final state vectors are packaged via serialization routines and sent to the back-end state service for long-term storage and consistency.

$$\sigma = s\left(b^{-1} + c^{-1} + x^{-1}\right) \tag{4.3}$$

The time to propagate state information (σ) from the client to the back-end services or vice versa is mainly dependent upon two variables: state size (*s*) and available bandwidth (*b*). Associated overheads dependent upon state size such as memory copy throughput (*c*) are a secondary factor in the total time for a state update. *x* denotes either state serialization or deserialization time depending upon if the state is being packed for update or unpacked for a lease to update state vectors at the client respectively.

4.4.2 Message Packing, Transfer and Binning

Under a master/worker PDES system, all generated messages must be packed and buffered before they are sent to the back-end service. Once messages are received by the back-end message storage service, the binning process delivers each message to the proper input queue of the destination work unit.

$$\mu_p = nm\left(b^{-1} + y^{-1} + i^{-1}\right) \tag{4.4}$$

Message transfer time to the back-end services (μ_p) is primarily determined by the number of messages (*n*), average message size (*m*) and the bandwidth available (*b*). The other variables include message serialization time (*y*) and time required to bin messages (*i*).

4.4.3 Message Unpacking and Client Delivery

Message delivery to the client incurs overhead captured in equation (4.5). Once a lease is given to a client, it contacts the appropriate message server and downloads all messages within the lease execution window.

$$\mu_u = nm\left(b^{-1} + y^{-1}\right) \tag{4.5}$$

The time required to transfer messages to the client for a lease (μ_u) is similar to message packing and return except for the absence of binning time and *y* denotes message deserialization time required to unpack messages on the client site.

4.4.4 Deterministic Lease Overhead

Each work unit lease incurs overhead, of which a portion is deterministic and a portion is non-deterministic. The non-deterministic portion is referred to as the deferred wait time where a client will enter a sleep state waiting for the next available work unit. There are a variety of reasons for this to occur including the number of workers exceeding the available work unit pool or some work units containing disproportionate amount of work. Equation (4.6) represents the deterministic portion of the lease overhead.

$$l = \max\left(\sigma_d, \mu_u\right) \tag{4.6}$$

Total deterministic lease overhead (*l*) can be calculated by taking the maximum of the time to download and unpack the state along with the time to deliver messages to the client and deserialize them. A maximum function is applied as these two independent operations are performed simultaneously by the client in two separate threads.

4.4.5 Finalization Overhead

Finalization overhead, in contrast to lease overhead, is completely deterministic. After the client completes the application simulation by reaching the end time of the lease window, the finalization phase begins. Finalization consists of performing the logical

inverse of a work unit lease: packaging state vectors through serialization methods and uploading both state vectors and messages to the back-end services.

$$f = \max\left(\sigma_u, \mu_p\right) \tag{4.7}$$

Again a maximum function is applied as the state update and message upload are performed simultaneously.

4.4.6 Total Deterministic Overhead

Total deterministic overhead as observed by the client per work unit lease can be expressed as the sum of the lease and finalization times as shown in equation (4.8).

$$o = l + f \tag{4.8}$$

$$o = \max\left(\sigma_d, \mu_u\right) + \max(\sigma_u, \mu_p) \tag{4.9}$$

Equation (4.9) is the expanded form of equation (4.8) clearly illustrating that overhead is dictated and is proportional to either state vector or aggregate message size. Thus, simulations that exhibit large state vectors will require optimizations for state transfer and storage. Simulations that send large numbers of messages will require optimized mechanisms to expedite delivery of messages. Moreover, the reduction of overhead through these techniques must be applied concurrently as the reduction of one overhead component can make that overhead time fall below the overhead of the other component. Applying complimentary techniques provides a more optimal reduction in overhead for the entire system.

4.4.7 A Model for Master/Worker Parallel Runtime

The previous equations apply to a per-lease cycle basis. There are multiple lease cycles in any non-trivial simulation consisting of more than one work unit with message passing. The total lease cycle time (T) can be computed as:

$$T = \alpha + o \tag{4.10}$$

The total time is simply the processor time due to application computation (α) for the lease and the deterministic overhead time (o). To compute the total runtime, an additional component is needed, the number of lease cycles (L):

$$L \approx \sum_{i=0}^{\text{All WU}} \frac{endtime}{MinETS_i} \tag{4.11}$$

The number of total lease cycles required to complete a simulation can be estimated by summing the number of time intervals (e.g., simulation endtime divided by the number of minimum emittable time stamp bounds) for each work unit in the system. This provides an upper bound (largest number of lease cycles). In an actual simulation, however, the number of lease cycles is dependent upon the work unit lease order. Thus L is represented as an approximation in equation (4.11). A work unit dependent upon another work unit may have already been processed allowing a larger time window than the minimal window. The total time required then simply becomes the product of T and L:

$$\omega = TL \tag{4.12}$$

The total sequential runtime (ω) is equivalent to running the simulation with a single client, as there would be no idle cycles present in the system. Work would always be available without delay immediately after a work unit is finalized and returned to the back-end services.

At best, only an estimation of total time can be given for parallelizing the simulation across many clients. Moreover, assumptions about the simulation environment must be made clear.

1. The number of clients available in the system must exceed the number of runnable work units in the system at all times.
2. Processor speeds and available bandwidth to each client are similar. While they do not have to be the same across every client, a large difference such as a dial-up link versus a gigabit Ethernet link would introduce inconsistencies into the model.
3. Each processor must be fully dedicated to the worker client.

The total parallelized run time (Ψ) can be expressed as:

$$\Psi \approx \left(T + d\right)\frac{L}{C} \tag{4.13}$$

This equation introduces deferred wait (d) overhead time and the number of clients in the system (C). Deferred wait refers to the amount of time a client is spent idling for a valid work unit to be leased to it. Deferred wait is a non-deterministic variable dependent upon PDES application characteristics such as lookahead and event granularity and non-PDES related effects such as processor speed, the number of available clients, and available bandwidth. Not all work unit leases are equal, as the amount of computation time may vary, the event generation and processing time may greatly vary along with the amount of state and messages that must be synchronized with the back-end services. The uneven processing between leases provides a non-exact estimation on total runtime.

In order to minimize the total parallel runtime, the total time per lease must be reduced, $T + d$. The reduction of one variable may affect the other as there may be a trade-off in certain PDES simulation models. For instance, attempting to minimize o

(which is a component of T) may directly impact the deferred wait time, d. The reduction of T requires the reduction of overhead components l and f. As shown in equation 4.7, these values are directly determined by the amount of state or aggregate amount of messages generated per lease cycle. Techniques such as caching may reduce l and f but can adversely affect d. For example, attempts to maximize cache hit ratios can lead to increased deferred wait time as the system may block and wait for clients with valid cache lines that are busy.

4.5 Scheduling Policies

Implementation of a caching system in a master/worker PDES system involves two separate sub-systems that must interoperate to provide effective mechanisms to reduce overheads consisting of state shuffling and delay times due to idle wait cycles. The first portion resides on the master side of the system where the scheduling and work unit reservation mechanisms must determine the best tradeoff between cache hit ratio and minimizing client latency and idle wait cycles for work unit leases. The second portion resides on the worker side where proper eviction and replacement strategies must be in place to maximize cache hit ratios when insufficient memory is available to cache fresh state vector blocks.

In a master/worker PDES system without a caching system, work units are scheduled on a first-come first-served basis where the operating principle is to minimize idle cycle wait time on clients. If a work unit is available, the work unit is matched to an idle client and immediately leased. In a master/worker PDES system that incorporates caches, the scheduling mechanism for releasing work units to clients must be reworked to

provide a balanced approach between leasing to clients that have valid cache lines and minimizing client idle cycle time.

4.5.1 The Rub: Minimizing Overhead vs. Minimizing Deferred Wait

A scheduling mechanism for a master/worker PDES caching system can be characterized as providing the best mix between work unit cache affinity and idle cycle wait time as shown in Figure 46. There exists a tradeoff between attempting to maintain maximal cache affinity and optimizing around the highest possible cache hit ratios over client idle cycle times.

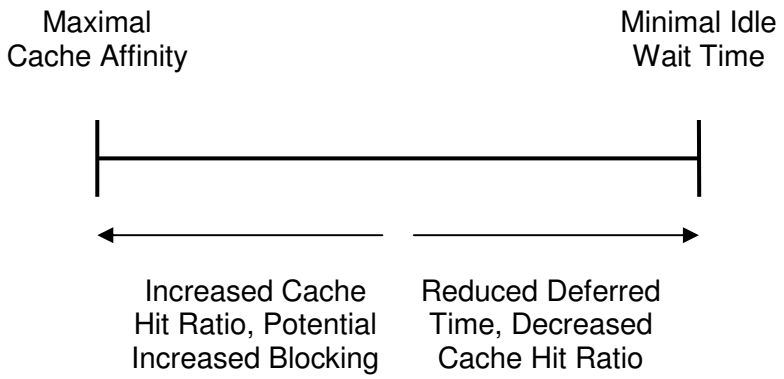

Figure 46: Tradeoff between Cache Affinity and Idle Wait Time

In a master/worker PDES system, a client may be actively processing a work unit but may also have the only valid cache line for a work unit that is ready to be processed immediately. A mechanism that preserves maximum cache affinity would reserve such a work unit for the currently busy client adversely affecting the idle wait cycle time of other available clients. On the other hand, a system which aims for minimizing idle cycle wait times of clients can potentially increase cache misses, incurring higher bandwidth and overhead costs.

A system that acknowledges that a balance between maintaining affinity when possible but considers that potential idle wait times can also reduce overall throughput of the system will provide a more optimal solution than a system at either extreme. Since PDES applications are varied in scope, determining where to place the "slider" in Figure 46 is impossible to determine a priori. Instead a system that gathers statistics and analyzes them at run time to fine tune the scheduling mechanism dynamically using minimal input from the simulation modeler is the most flexible approach. The following sections detail proposed scheduling policies based on a caching system for master/worker PDES. Scheduling policy equations compute a scheduling priority value, Φ, which is used by the back-end proxy service to order work units in increasing order, where highest work unit Φ values are scheduled to the requesting client.

4.5.2 Maximum Cache Affinity (MCA) Scheduling

This scheduling policy attempts to maintain the maximum affinity for valid caches at the clients. Assuming a sufficient number of clients are available for the number of work units in the system, the scheduler will simply match idle work units with clients that have valid cache entries, preferring idle clients over those that are busy. However, a reservation will still be made even if the client is busy.

Although this scheduling policy can maintain maximal cache hit ratios (given enough clients in the system), reserving a work unit onto an already busy client while other clients are idle leads to increased deferred wait time where clients are essentially blocked from processing available work. A PDES code that exhibits very large states for which there is insufficient available bandwidth, the tradeoff in increased blocking and wait time may be worth the exchange for a cache hit.

The scheduling priority of a work unit (Φ) can be characterized as a function of the runnable status of work units and other prioritizing factors such as valid cache lines.

$$\Phi = \rho\chi \qquad \text{if } \frac{C}{W} \geq 1 \tag{4.14}$$

$$\Phi = \rho + \chi \qquad \text{if } \frac{C}{W} < 1 \tag{4.15}$$

Equations (4.14) and (4.15) prioritize work units based on its runnable status (ρ), whether the work unit can advance ρ = 1 or cannot advance ρ = 0 and if there is a valid cached state at the requesting client (χ). *C* denotes the number of clients while *W* specifies the number of total work units in the system. Exceptions to these equations are made during the initial lease phases when all cache lines are invalid. Equation (4.14) ensures that if enough clients exist in the system with valid cache lines, such work units are leased to matching client caches. Equation (4.15) prioritizes runnable work units to clients with valid cache states when not enough clients exist in the system.

4.5.3 Minimum Idle Wait (MIW) Scheduling

A scheduling policy that attempts to minimize client idle wait times does so at the expense of potential cache hits on valid cache entries on busy clients. The premise behind this scheduling policy is to always ensure that available work units are run on any available client, regardless of a valid cache line on a busy client. If idle clients contain valid cache states, these clients are matched first over idle clients that would invoke cache misses. If the PDES application exhibits small overheads in transmission time due to state vector updating, this policy may provide the largest benefit as the value of a cache hit is significantly reduced over a reserved work unit for a busy client.

$$\Phi = \rho\chi(1-\beta) \tag{4.16}$$

Equation (4.16) prioritizes work unit leases based on an additional conditional of the busy status (β) of the client holding the valid cache line where a busy state is β = 1.

4.5.4 Idle Wait Time Aware Cache Affinity (IWTACA) Scheduling

In a balanced approach, the scheduling policy dynamically attempts to prioritize cache affinity but is aware that prolonged idle client wait times can negatively impact the throughput rate of the system. This mechanism employs runtime statistics to track application time and state vector sizes and associated transmission times. Given these statistics, the master service can determine if it is worthwhile to reserve a work unit for a client which has valid cached data but is busy over leasing the work unit to an idle client which would invoke a cache miss.

$$\Phi = \rho\left(\frac{\sigma\chi}{\alpha} - \frac{\alpha\beta}{\sigma}\right) \tag{4.17}$$

A balanced approach to work unit leases involves analyzing run time statistics such as average application runtime (α) and average state transmission time (σ). In equation (4.17), the presence of a valid cache line is modified by the ratio between state transmission time and application time, while this ratio is inverted and applied to the busy status of a client with a valid cache state. These values are summed together to give a prioritizing number for a possible work unit lease.

4.5.5 Weighted Fan-Out (WFO) Scheduling

A master/worker PDES system can leverage key properties from the application to provide improved scheduling priorities on work units. Although an idle wait time

aware cache affinity priority system can dynamically adjust work unit scheduling at run time, it is only limited to separating work units into runnable and non-runnable states. By utilizing PDES domain-specific properties such as connectivity graphs and lookahead, the scheduler can maximize work unit activity while attempting to ensure high cache hit ratios. We introduce a variable that measures the "impact" of a work unit by counting the number of output links and the lookahead values. This value, η_i, is the weighted fan-out value for work unit i:

$$\eta_i = 1 + \frac{\Lambda}{\sum_{j=0}^{\Lambda} \lambda_j} \tag{4.18}$$

In equation (4.18), Λ represents the number of output links from work unit i to other connected work units and λ_j is the lookahead of link j at work unit i. The number of output links is divided by the total lookahead over these links to provide a weighted fan out value. Smaller lookahead values decrease the minimum emittable timestamp (MinETS) values used in window calculations, thus is reflected as an inverse relationship making η larger for smaller values of lookahead.

Equation (4.17) can be modified to incorporate the weighted fan-out metric in the scheduling priority value:

$$\Phi = \rho\eta\left(\frac{\sigma\chi}{\alpha} - \frac{\alpha\beta}{\sigma}\right) \tag{4.19}$$

Larger weighted fan-out values increase the scheduling priority of the work unit as desired. Equation (4.19) provides the scheduler with lookahead and connectivity information to possibly schedule work units which may lie in the critical path of the simulation.

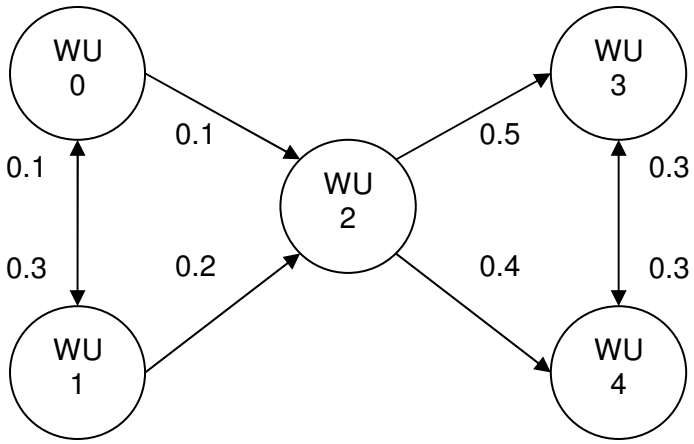

Figure 47: Example Work Unit Connectivity

Figure 47 shows a sample connectivity layout for a sample simulation (e.g., a network simulation) where the delays between links represent the lookahead values. Messages flow from left to right in this simulation, thus work units 2, 3, and 4 cannot arbitrarily proceed into the future and are restrained by forward progress of work unit 0 and 1 dictated by LBTS and LCC principles. A scheduler that does not prioritize based on PDES specific properties such as lookahead and connectivity may lease work units in a non-optimal fashion. A lookahead and connectivity aware scheduler, in addition to optimizing for cache affinity and minimizing idle wait cycles, attempts to provide the maximum amount of concurrent work and largest time windows for clients.

Table 6: Weighted Fan-Out and Priority

Work Unit	η	Relative Priority (0 = highest)
0	6.0	1
1	7.667	0
2	3.222	4
3	3.5	3
4	3.867	2

Table 7: Arbitrary Leasing

Work Unit	Sim Time	Lease 1	Lease 2	Next Advance
0	0.1	HOLD	HOLD	0.5
1	0.3	HOLD	0.4	NA
2	0.1	HOLD	0.2	NA
3	0.3	0.6	HOLD	0.7
4	0.3	0.5	HOLD	0.6

Table 8: Weighted Fan-Out Priority Leasing

Work Unit	Sim Time	Lease 1	Lease 2	Next Advance
0	0.1	0.4	HOLD	0.8
1	0.3	0.4	HOLD	0.7
2	0.1	HOLD	HOLD	0.5
3	0.3	HOLD	0.6	NA
4	0.3	HOLD	0.5	NA

In this example simulation, assume that there are two clients available for processing work units. Table 6 shows the calculated weighted fan-out and relative priority ranking between work units where 0 is the highest and 4 is the lowest. For simplicity, the caching, busy status, and runtime statistics portion of equation 4.19 are ignored.

Table 7 shows an arbitrary lease scenario. Assume the first round of initial leases for all work units have been processed. In the next phase, the clients are assigned work units 3 and 4 to advance forward in simulation time, while work units 0, 1, and 2 are held back. The second round of leases, work units 1 and 2 are released to the clients for

processing while work units 0, 3, and 4 are held. For the next round of processing, work unit 0, 3, and 4 have valid processable time windows (NA denotes not advanceable).

Table 8 shows a leasing scheme in which the scheduler is aware of lookahead and connectivity and attempts to optimize using this information. In the first lease round, work units 0 and 1 are processed first followed by work units 0 and 2 in the second set of leases. In the next round of processing, work units 1, 2, 3, and 4 are all available for processing giving the system more concurrency with less idle wait time for clients.

A weighted fan-out scheduling policy exploiting lookahead and connectivity graphs avails more simulated time for leasing and the progress of the simulation is further than an arbitrary scenario. In the arbitrary lease scenario, the next advance window lengths for work unit 0, 3, and 4 are 0.4, 0.1, and 0.1 respectively combining for a total of 0.6 time units of processable concurrent simulation time. With a scheduler taking into consideration lookahead and connectivity, work units 0, 1, and 2 have execution window lengths of 0.4, 0.3, and 0.4 respectively combining for a total of 1.1 time units of processable concurrent simulation time. Additionally, at the next advance time, the simulation will be further ahead for the weighted fan-out priority: 0.8, 0.7, 0.5, 0.6, and 0.5 compared to the arbitrary lease scenario of 0.5, 0.4, 0.2, 0.7, and 0.6 for work units 0 through 4.

4.5.6 Earliest End Time First (EETF) Scheduling

In Time Warp systems, Lowest Timestamp First (LTF) schemes are often used to schedule events that are the least likely to be rolled back and create messages that must be unsent. Time information similar to scheduling based on the minimum TSO event can be utilized in an optimized cache-aware scheduling scheme. A conservative execution

that acknowledges earliest end time of a time window can reduce the amount of blocking in the system. Unfortunately, LTF schemes do not directly apply in conservatively synchronized master/worker systems as the atomic unit to determine what can or cannot be run along with the run length is not a singular event, but is instead a time window.

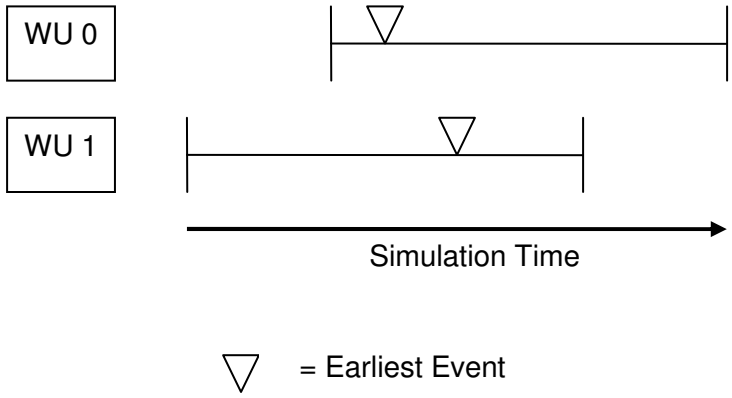

Figure 48: Example Time Window Comparison

Figure 48 shows a possible scenario where the earliest timestamped event does not necessarily coincide with the earliest end time of a time window in the system. Work units that depend upon work unit 1 advancing may block for extended periods of time if such advanceable work units are not prioritized. The guarantee for any other work unit which has work unit 1 as an input connection can only be the end time of the time window, thus the amount of blocking in the system can be minimized by ensuring such work units are leased first.

$$\tau_i = \frac{E - \delta_i}{E} \qquad \text{if} \quad \delta > 0 \tag{4.20}$$

$$\Phi = \rho\tau\left(\frac{\sigma\chi}{\alpha} - \frac{\alpha\beta}{\sigma}\right) \tag{4.21}$$

In equation (4.20), τ defines the relative fraction of simulation time remaining for work unit i. δ represents the minimum emittable time stamp for all input links (e.g., the effective simulation end time for the next lease window) to work unit i and E denotes the simulation end time. Larger τ values represent work units with earlier simulation end times. This value is incorporated into the IWTACA priority scheme which adaptively re-orders work unit priority for scheduling based on window end time information yielding equation (4.21).

4.5.7 Weighted Fan-Out and Earliest End Time First Scheduling

The weighted fan-out and earliest end time first (WFO+EETF) scheduling policy attempts to utilize all available information available including runtime statistics, valid cache block availability, client busy status, along with PDES-specific information such as lookahead, connectivity, and lease windows.

$$\Phi = \rho\eta\tau\left(\frac{\sigma\chi}{\alpha} - \frac{\alpha\beta}{\sigma}\right) \qquad (4.22)$$

Equation (4.22) represents work unit to client priority for a WFO+EETF scheme where the IWTACA policy is augmented with the weighted fan-out (η) and earliest end time first (τ) values.

4.6 Performance Study

An empirical study was performed to assess the proposed techniques for overhead reduction in a master/worker PDES system. For these tests, the queuing network and hybrid shock codes were used as in previous experiments. The only change was the queuing network simulation was modified to allow variable (non-uniform) lookaheads

between work units. To provide a comparison point against a traditional parallel simulation system, some tests for the hybrid shock application were compared against a native version under μsik [94]. μsik is a parallel and distributed simulation framework accommodating a wide variety of optimistically- or conservatively-synchronized applications through a micro-kernel approach. μsik utilizes the libSynk [95] library for distributed time management which has shown to be highly scalable across many applications. In comparison tests, we measure end-to-end performance: the total wall clock time from the beginning of the simulation to the end of the simulation. For fair comparisons, μsik was not run with shared memory enabled; inter-processor communication used TCP transport.

The following experiments were performed on a set of nodes containing dual 3.2GHz Intel Xeon processors with 6GB of memory per node. Each node was running a RedHat GNU/Linux 2.6.9 64-bit kernel. Nodes were connected through Fast Ethernet links. All software was compiled using gcc 3.4.6 and using the -O3 optimization flag. POCO version 1.3.2 was used for Aurora tests.

4.6.1 Work Unit Granularity Selection

Determining how many work units are in the system given an anticipated worker pool size is important for delivering the best possible performance without introducing artificial overheads and constraints. Therefore, before any performance tests examining the proposed mechanisms are performed, the correct work unit granularity size must be selected to ensure that sufficient work exists for the worker pool. The following tests scale simulations by varying the work unit size. For the torus queuing network simulation, the number of nodes is fixed, but the amount of total computation increases as

the work unit count increases as jobs are generated on a per work unit basis. For the hybrid shock scenarios, the entire problem size is fixed, thus as the number of work units increases, the amount of computation per work unit decreases.

The first test is a queuing network simulation with uniform lookahead between work units. The torus network is configured as 1.44 million nodes as a 1200x1200 grid. Lookahead between work units is set at 0.1 seconds with internal lookahead within each work unit between nodes set at 0.0001 seconds. A small lookahead was chosen to ensure that enough computation and event processing was performed during each work unit lease. 100,000 jobs are generated within each work unit and 20,000 jobs are generated for nodes external to the work unit. The mean job service time is 0.2 seconds. Caching is enabled for these tests with the default MIW scheduling policy. These torus queuing network simulations are run across 32 processors and 4 state and message servers are instantiated on the back-end system.

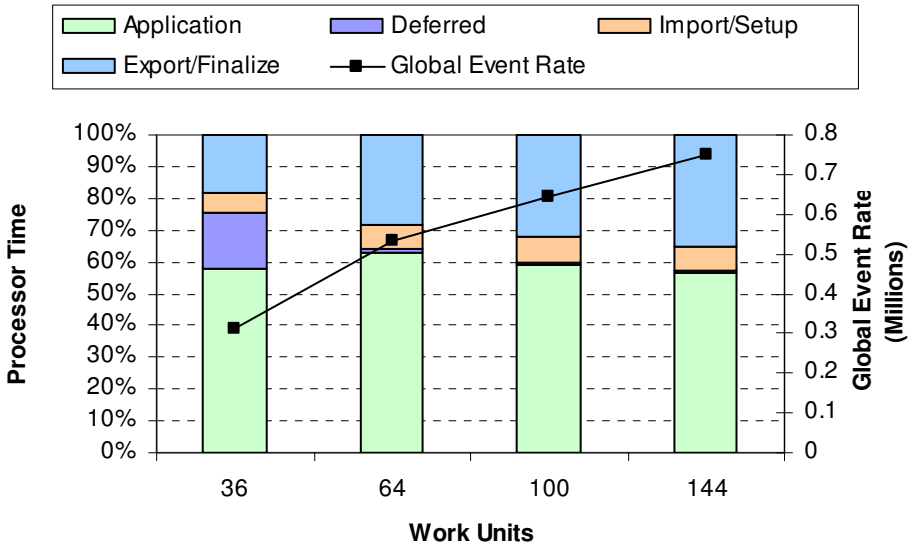

Figure 49: Uniform Lookahead

The striking result from the uniform lookahead test shown in Figure 49 is the amount of deferred wait time incurred when 36 work units compose the worker pool. Deferred wait time is nearly eliminated when the worker pool is twice the number of workers in the system. This suggests that merely having a worker pool that matches the anticipated number of workers is insufficient leading to blocking time for some clients.

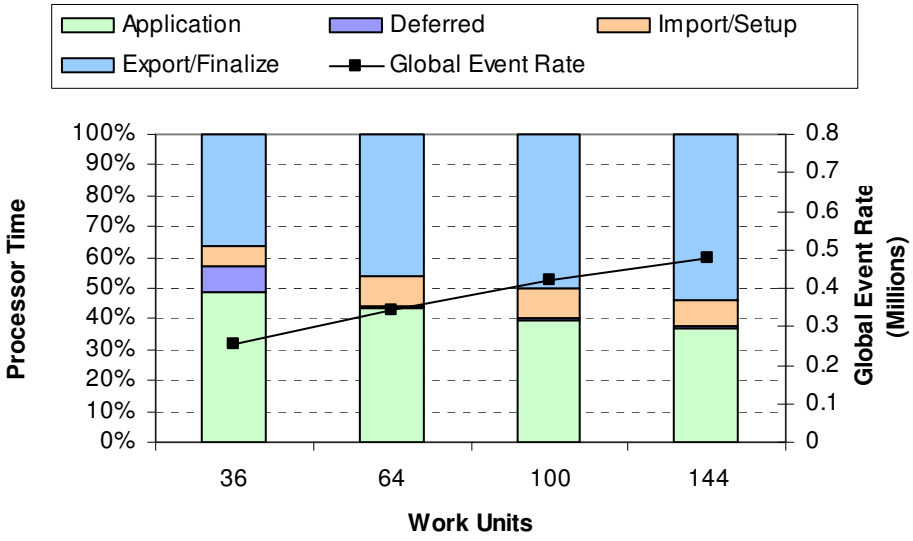

Figure 50: Non-Uniform Lookahead

The non-uniform lookahead test randomly varies the amount of lookahead upon creation of the network. The inter-work unit lookahead values vary from 0.05 to 0.1 seconds. Lookaheads within each work unit vary from 0.0001 to 0.01 seconds. As shown in Figure 50, lower overall performance is observed due to less concurrency in the system due to variable and smaller execution time windows. Similar to the uniform lookahead case, at least twice the number of work units as workers is needed to nearly eliminate deferred wait time due to an insufficient amount of available work.

To test simulations that do not increase the amount of overall computation in the system as the partitions are increased, the hybrid shock simulation is used. This simulation of 1.2 million ions exhibits a uniform lookahead of 0.11 time units between partitions with simulation parameters of 1200 cells, 1000 initial ions per cell, and a cell width of 0.00025. This simulation is run across 8 processors with 4 state and message back-end services. Work units are scheduled with the default MIW policy.

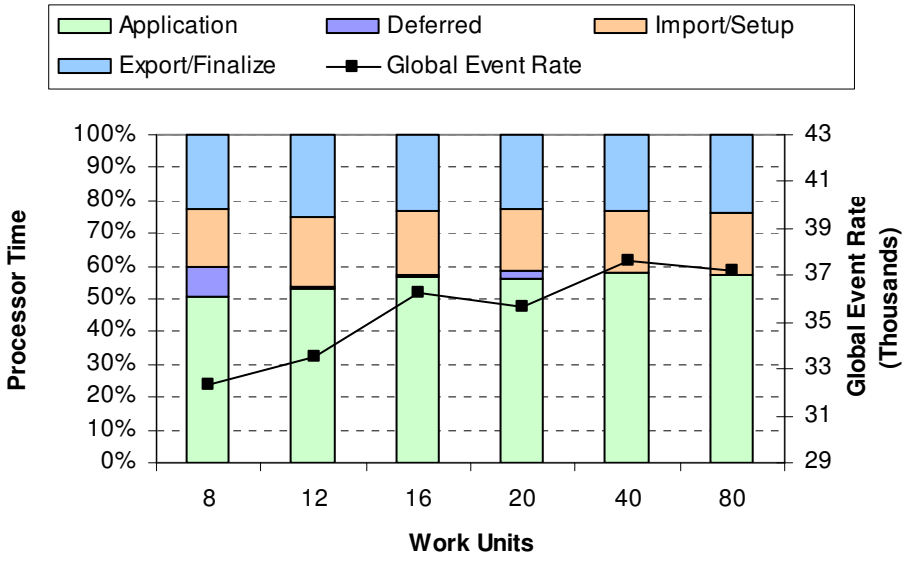

Figure 51: Hybrid Shock with Caching Disabled

Figure 51 shows similar trends to those exhibited by the queuing network. Simply matching the number of work units with the number of clients introduces artificial deferred wait time overheads. By increasing the number of work units to at least 150% of the anticipated worker pool reduces deferred wait times to a negligible level.

4.6.2 Work Unit Caching

The performance impact of enabling client-side caching is evaluated by comparing baseline scenarios which have no optimizations. The following test is exactly the same scenario as shown in Figure 51, except caching is enabled.

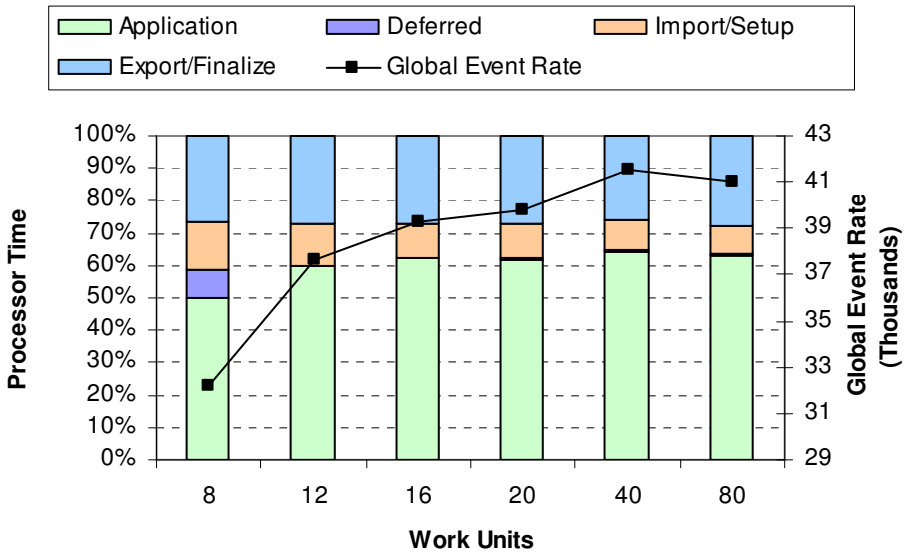

Figure 52: Hybrid Shock with Caching Enabled

With caching enabled, the amount of import and setup time is decreased, leading to higher throughput and increased global event rate as shown in Figure 52. Again, these results suggest that there should be at least 150% the number of work units as there are clients in the worker pool even under a cached system.

To compare against typical traditional PDES system performance, this 8 processor run is compared against a μsik run with the same parameters simulating 1.2 million ions. In order to provide an even comparison of Aurora with μsik, the end-to-end time to complete the simulation performance metric is used. In other words, the total wall clock time from the time the simulation starts to when the simulation completes (e.g.,

the last node or client finishes computation). Percentage of μsik performance denotes the proportional amount of time that Aurora takes in comparison to μsik. In the following Figure 53, μsik ran the simulation in 119.044 seconds. Utilizing 8 work units, the baseline (caching disabled) Aurora run took 228.698 seconds, or 52.05% of μsik performance.

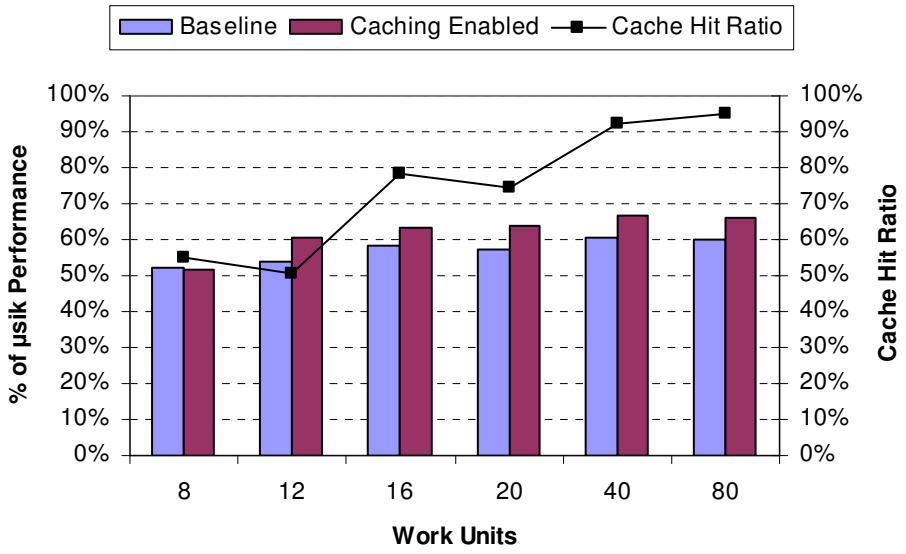

Figure 53: Work Unit Scaling, Cache Hit Ratio and Comparison with μsik

Under a MIW scheduling policy, the more work units that are available in the system, the higher the likelihood of a valid cache block existing on a client that can immediately be matched during a work unit request. The 40 work unit partitions exhibited the best performance for this scenario under Aurora, where 60.52% of the μsik performance was obtained for the caching disabled run and 66.83% of the μsik performance was obtained for the caching enabled run. It is important to note, that this is a small scale test where these results only capture the client-side caching optimization as

all other optimizations such as state pipelined updating, pro-active message updates, and other scheduling policies were disabled.

To test larger simulations sizes across more processors the non-uniform torus queuing network was utilized simulating 2.25 million nodes on a 1500x1500 grid. This simulation was partitioned into 100 work units of 150x150 sub-networks with variable inter-work unit lookahead between 0.05 and 0.1 seconds, internal work unit lookahead between 0.0001 and 0.001 seconds, and a mean service time of 0.1 seconds. 150,000 local packets and 45,000 remote packets were generated. This simulation was run across 32 worker processors with 6 state and message services.

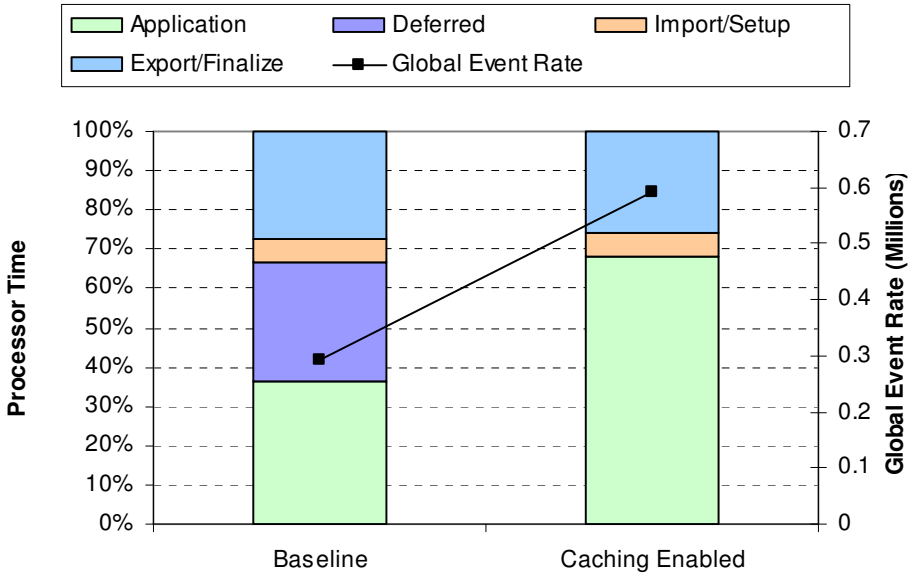

Figure 54: Effect of Caching on a Non-Uniform Torus Queuing Network

With the default MIW scheduling policy, this simulation with caching enabled exhibited a cache hit ratio of 66.42%. As shown in Figure 54, performance is dramatically improved by allowing clients to cache work units. The end-to-end performance of the baseline run was 543.73 seconds, while the caching enabled run took

328.19 seconds exhibiting a cache hit ratio of 66.42% and a speedup of 1.66. The cache-enabled run bypassed 2970 MB of state downloads across all clients (92.81 MB per client) resulting in a bandwidth savings of 69.85%.

4.6.3 State Updates

Enabling a caching protocol provides overhead reduction during the import/setup phase by allowing the client to bypass state downloads when a valid cache block is detected. However, to ensure consistency between the clients and the back-end services the state must be synchronized with the state storage service so that other clients may download the state if the current worker fails or disengages from the simulation. By employing a pipelined state updating mechanism that overlaps communication costs with application computation time, the export and finalization phase can be expedited.

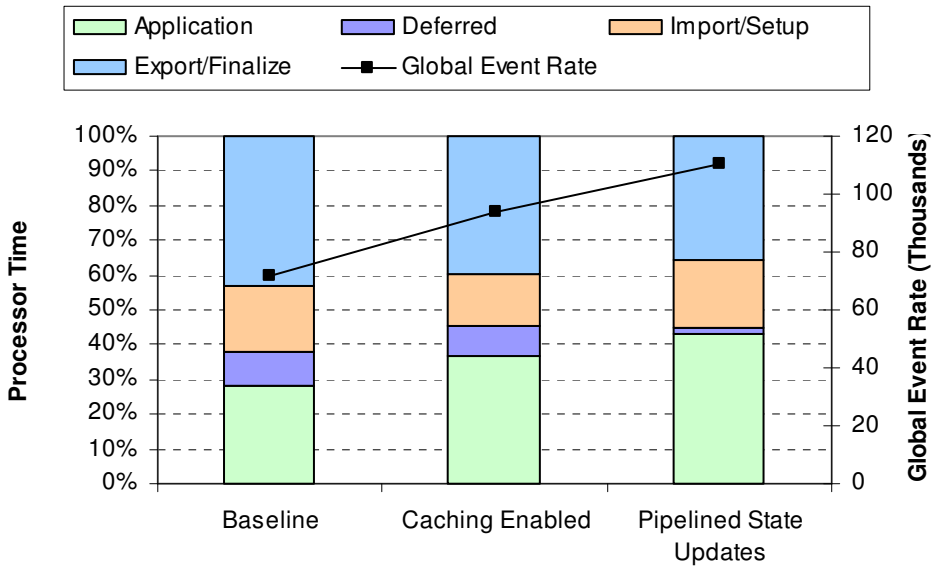

Figure 55: Hybrid Shock with Pipelined State Updates

Figure 55 shows performance trends of a 32 client hybrid shock run containing 3.2 million ions over 3200 cells partitioned as 80 work units. With caching enabled, the

simulation exhibits a cache hit ratio of 67%, a bandwidth savings of 67% (2870.7 MB of state downloads bypassed) and a speedup of 1.31 over the baseline run. The pipelined state updates test showed a cache hit ratio of 67.75%, a bandwidth savings of 67.76% (2902.9 MB of state downloads bypassed) and a speedup of 1.54 compared to the baseline run. The pipelined state update mechanism reduced export/finalize time from the base cache-enabled run from 83.64 seconds to 63.85 seconds on average across all clients.

Pipelined state updates may not offer performance improvements in all cases. In certain cases, minimal performance gains or performance degradation can result if not enough application computation time exists given the size of the state.

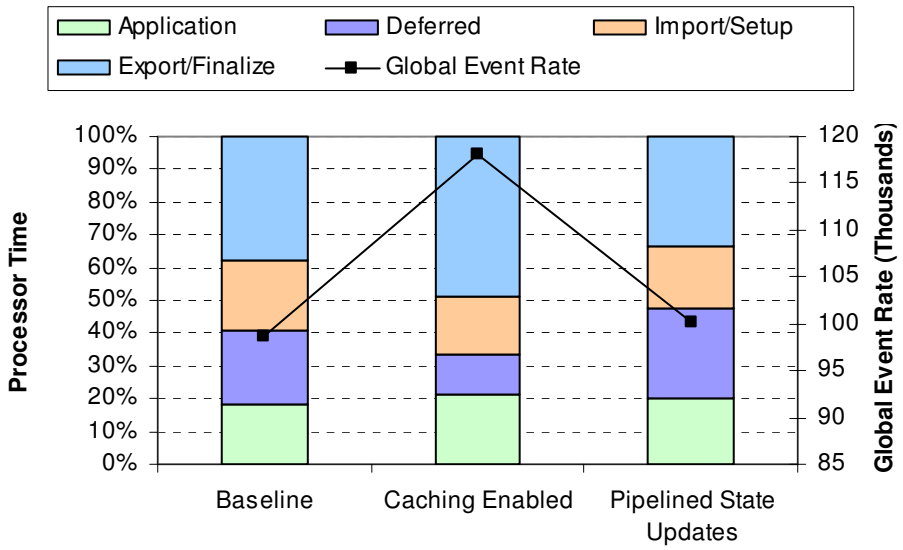

Figure 56: Insufficient Masking Time

Figure 56 shows a 70 client (processor) run utilizing a 160 work unit pool for a hybrid shock simulation with identical parameters as the previous test. In contrast to the previous 32 client run, this test has significantly less computation per work unit, thus

resulting in less masking time for pipelined state updates. The pipelined state updates increase the deferred wait time in the system as state updates are delayed from when work units are checked back-in. Clients may also block for additional time during a work unit request phase if a state update is in progress after a lease is approved. The state update must complete before the work unit begins on the client, further increasing the deferred wait time as shown in Figure 56.

4.6.4 Eviction and Replacement Policies

Clients may encounter memory pressure forcing eviction and replacement of pre-existing cached states stored in the state history vector. The following tests examine proposed replacement policies. The first benchmark uses the torus queuing network distributed across 32 clients, with all simulation parameters maintained from the previous experiments.

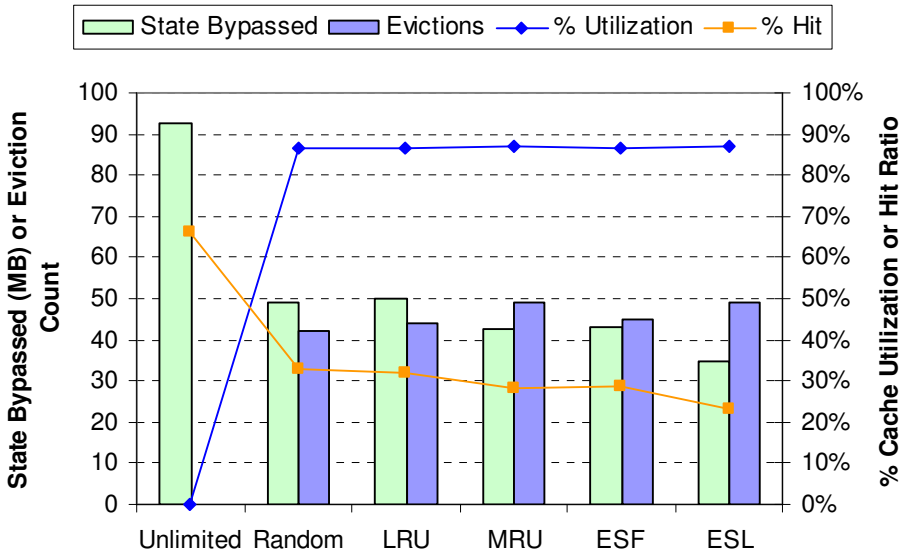

Figure 57: Replacement Policies and Queuing Network Simulation

Figure 57 shows the results from a queuing network test where each client is only allocated 10 MB of total cache space. Each work unit packed state is approximately 2.15 MB, thus each client cache can only hold approximately 4 to 5 states. None of the proposed replacement policies show gains over a random replacement policy suggesting that replacement policies need assistance from the proxy service on which cache blocks to evict.

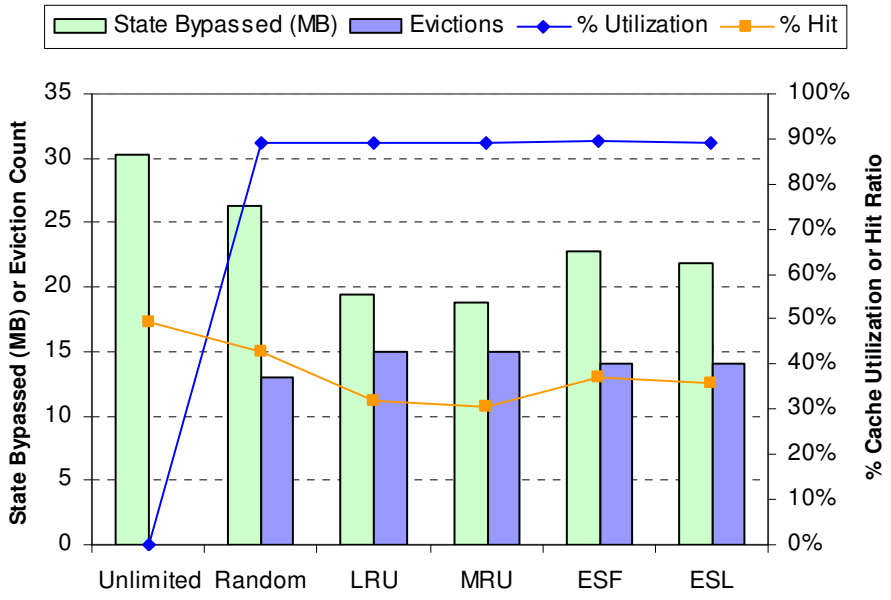

Figure 58: Replacement Policies and Hybrid Shock Simulation

For the 70 client hybrid shock test, Figure 58 shows higher performance for the ESF and ESL replacement strategies over LRU and MRU but still fall below the performance of the random replacement policy. Similar to the torus queuing network results, a different approach to eviction and replacement is needed in master/worker PDES. Specifically, a protocol is needed where the client can either query the proxy service on work units that may be runnable in the immediate future or piggybacking this information on control messages of other requests or acknowledgements.

4.6.5 Message Updates

Allowing clients to update messages in smaller chunks as the simulation generates them during the application computation phase can potentially lead to smaller export/finalization times. All performance tests have state pipeline updating disabled to only test the effect of pro-active message mechanisms. The first test examines the non-uniform queuing network using the same parameters as the previous 32 client runs.

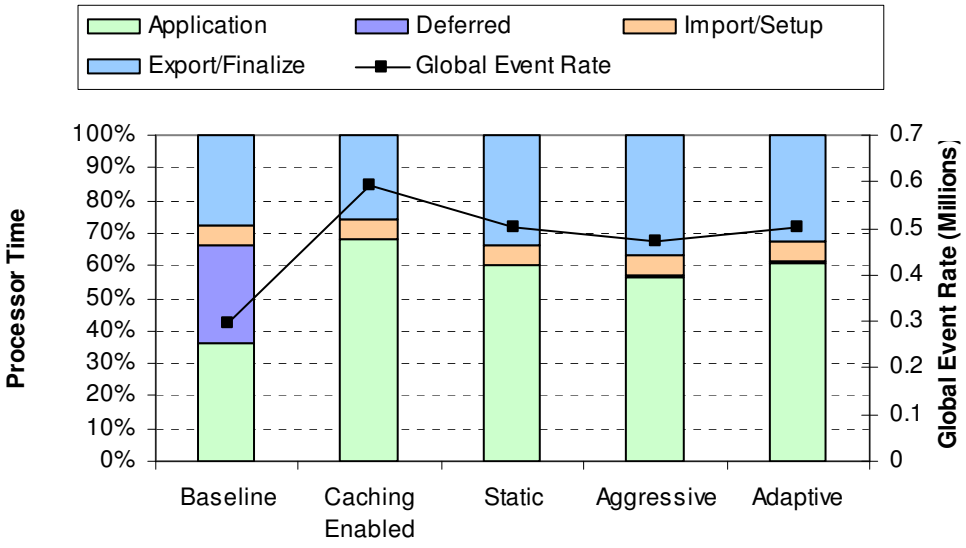

Figure 59: Queuing Network Variable Message Updates

Figure 59 shows performance trends with different message updating schemes. Between the three mechanisms, the aggressive approach performs the worst. However, interestingly, all three schemes are outperformed by a fully buffered approach where all messages are buffered and sent during the finalization phase. Observed data shows more than 34 million inter-work unit messages were generated and each work unit generating approximately 1.07 million messages. The reason for the slowdown with the pro-active message sending approaches point to the message services bogged down in constant

updates reducing the responsiveness of other requests such as message check-outs by other clients.

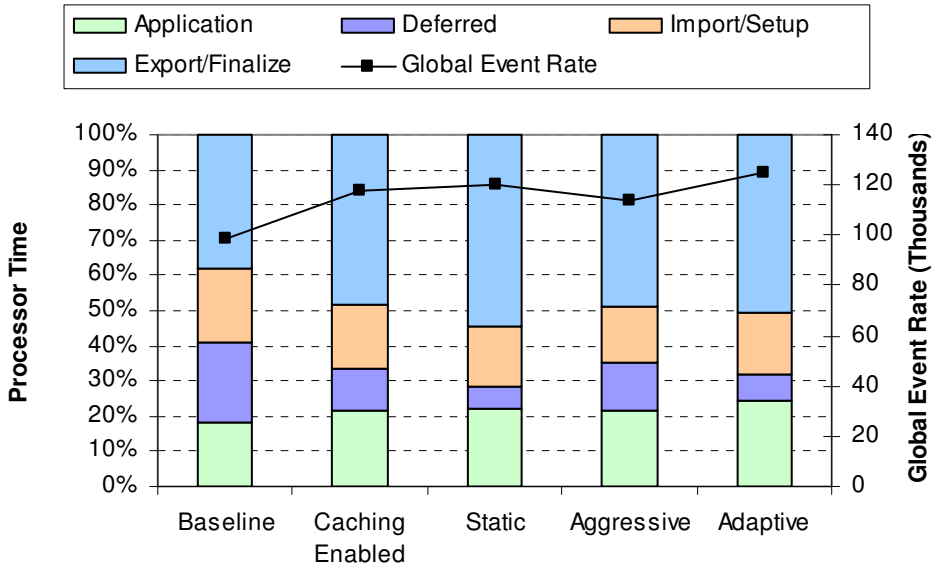

Figure 60: Hybrid Shock Variable Message Updates

The next performance test involves a 70 client hybrid shock run with the same parameters for a 3.2 million ion simulation as previous tests. In contrast to the queuing network run, this simulation generated 12.1 million fewer inter-work unit messages across all clients contributing to less pressure on the message services. This is reflected in Figure 60 where the performance of the adaptive message update algorithm shows improvement over the standard fully buffered method exhibiting a slight gain of 5.5% in the global event rate. The aggressive scheme performs the worst, providing more empirical evidence that overly pro-active message update schemes can prove to be a detriment to performance by increasing the load on the back-end message services and increasing latency of other requests.

4.6.6 Scheduling Policies

The following test examines proposed scheduling policies under a computationally intense hybrid shock simulation. The hybrid shock simulation is configured as 7.5 million ions arranged as 5000 cells and 1500 initial ions per cell. Other simulation parameters remain the same and the program is distributed across a pool of 32 clients. Caching is enabled with unlimited cache size and other optimizations disabled.

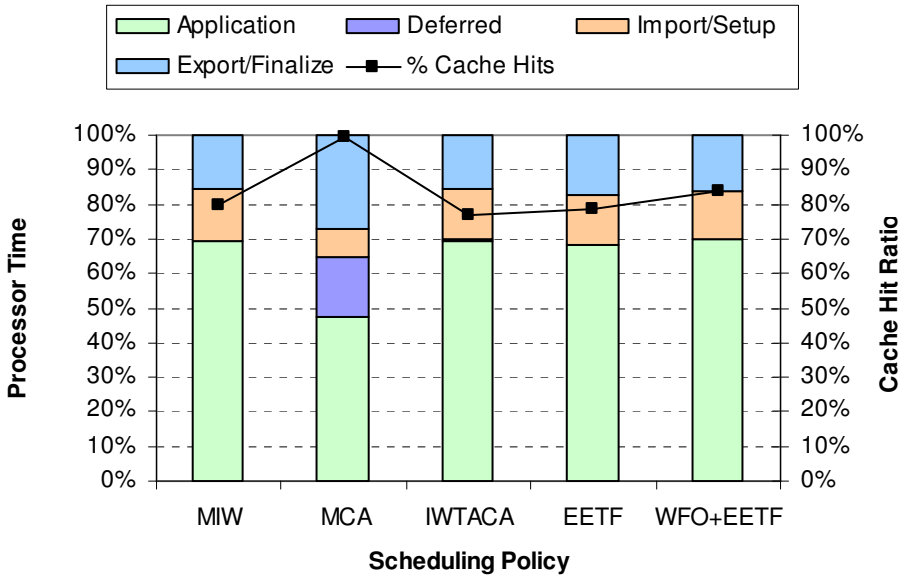

Figure 61: Scheduling Policy Performance under Hybrid Shock

Performance of each scheduling scheme is shown in Figure 61. The MCA policy exhibits the highest cache hit ratio at 99.3% along with the highest bandwidth savings avoiding 9962.12 MB of state downloads across all clients. However, the MCA scheme exchanges a high cache hit ratio with increased deferred wait times. The cost of this high cache hit ratio degrades the performance of the simulation such that this policy exhibits the worst global event rate out of all scheduling schemes at 72,616.1 events per second.

The WFO+EETF scheme provides the highest performance slightly edging out the MIW scheme at 123,051.8 global events per second.

4.6.7 Combined Optimizations

Individually, some overhead reduction techniques have shown to provide large boosts to performance while others have provided minimal gains or in some cases degraded performance. To better understand the impact of all overhead reduction techniques functioning in concert, the hybrid shock application is run Aurora and compared against μsik. The following hybrid shock scenario is configured as 7.5 million ions arranged in 5000 cells with 1500 initial ions per cell. There are 6 state and message back-end services. The number of Aurora clients matches the number of processors of the μsik run it is compared against. For example, the 8 Aurora client test represents a comparsion against a μsik run across 8 processors.

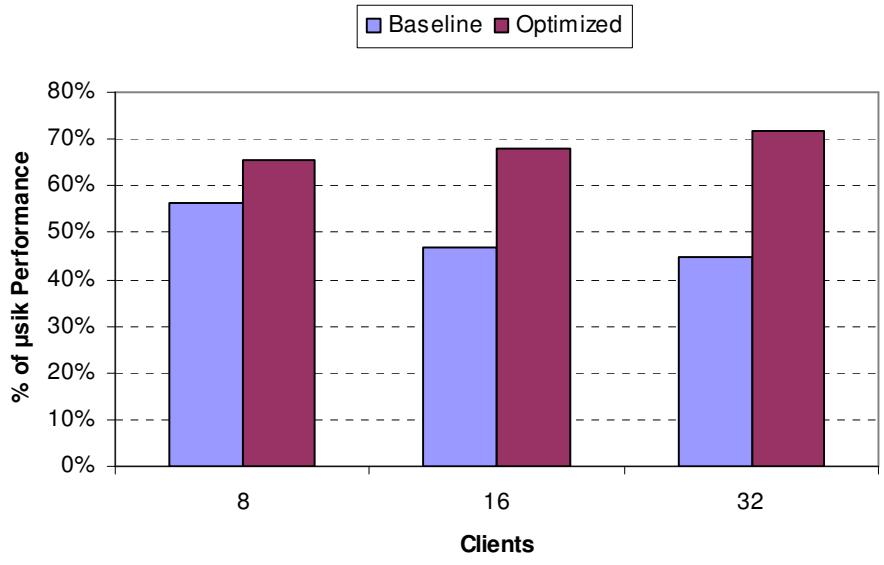

Figure 62: Hybrid Shock Optimized Performance

For this test, enabled overhead reduction techniques include client-side caching with "unlimited" cache space, pipelined state updates, aggressive message updates, and employing the WFO+EETF scheduling policy. Figure 62 shows the performance delta of Aurora under unoptimized baseline and fully optimized conditions. As the simulation scales with increasing numbers of clients, the unoptimized Aurora run performance gap widens from 55.28% of μsik at 8 workers to 44.59% of μsik at 32 workers. Interestingly, for the optimized Aurora runs, the performance gap shrinks as the number of clients is increased from 65.68% of μsik at 8 workers to 71.71% of μsik at 32 workers. At 32 clients, the overhead reduction techniques have recovered 27.12% of μsik performance.

With the flexibility and additional robustness that master/worker PDES brings to loosely coupled distributed computing infrastructures, the performance price is less than 30% for this hybrid shock simulation compared to a traditional simulation framework such as μsik. The Hybrid Shock application tested exhibits fairly large state vector imports and exports along with a sizable number of messages generated. Under PDES codes more conducive to the master/worker paradigm, the difference in performance from traditional PDES systems would be far less, providing even more incentive for utilizing a master/worker architecture. A master/worker PDES system offers the potential to capture idle-cycles of desktops and laptops that would otherwise be wasted. Given such a small performance gap now possible through these optimizations, a master/worker PDES system is a viable alternative to conventional systems while providing semi-automated load balancing, integrated fault tolerance, and portable simulations across heterogeneous machine architectures.

4.7 Conclusions

The barrier to acceptance of alternative PDES systems such as a master/worker system harnessing loosely coupled resources is the perception of low performance. Indeed, a master/worker PDES system with no optimizations can perform badly under certain simulation scenarios such as codes that exhibit large state vectors or those that generate large numbers of messages relative to the amount of computation that is performed. However, with certain optimizations that attempt to shift master/worker behavior to mimic traditional PDES systems, the performance gap can be reduced significantly.

Four overhead reduction techniques were presented and evaluated. First, arguably the most important optimization is work unit caching. The ability to store state vectors in a state history for future use avoiding download time, bandwidth consumption, and blocking time can provide large reductions in deferred wait and import/setup times. Second, a protocol for delaying the update of states through a pipelined updating mechanism allows work unit returns to be processed without blocking on large state updates. Other work units that are blocking due to a work unit dependency may be leased quickly with a faster consistency convergence on the work unit that has to be checked back in before the dependent work unit is leased. However, simulations that are not computationally intense may not provide enough masking time for pipelined state updates. Additionally, eviction and replacement policies must be in place for clients that are under memory pressure and must evict cache blocks for new states. Experimental data indicates that further investigation is required, specifically a protocol that provides the client with assistance in choosing the correct state to evict for replacement. Third,

protocols for pro-active message updating were proposed. Instead of buffering all messages until the finalization and export phase of a client, messages are sent during the application computation phase. Finally, scheduling policies were proposed that act on certain information such as valid cache lines, lookahead and connectivity information, and runtime statistics.

With these optimizations working together, the performance gap between master/worker PDES and traditional PDES was reduced significantly. Under a real-world simulation application, the performance difference was reduced to 29.29% of a high performance traditional simulation framework, μsik. While it is understood that a loosely coupled master/worker system will not provide the absolute fastest performance, we have shown that with proper overhead reduction techniques the performance of master/worker PDES systems is acceptable given the other advantages a master/worker PDES system offers.

CHAPTER 5

OPTIMISM AND MASTER/WORKER PARALLEL DISCRETE EVENT SIMULATION

Optimistic synchronization is attractive in this paradigm because it allows greater flexibility and concurrency for workers who can process events beyond those that are guaranteed to be safe for execution. This can improve the performance of simulations that are overly conservative by reducing the amount of master/worker communication. Further, due to the inherent volatility of workers in public resource computing and desktop grid environments, one must assume computations may be lost due to worker failure, network disconnects, or simply suspension of the process in order to complete other higher priority work at the client site. A system that can utilize optimistic execution can exploit this fact by allowing the lease of computations (e.g., an atomic unit of work released by the master service to a worker) that may potentially be overly optimistic and discard any returned results that are later rolled back. Moreover, it is well known that no single synchronization mechanism yields the best performance across all parallel discrete event simulation applications [1]. In particular, it is widely recognized that PDES codes that exhibit very small lookahead as well as those applications where the lookahead cannot be determined a priori are generally poorly suited for conservative synchronization techniques, suggesting the use of optimistic synchronization. However, traditional approaches to optimistic execution are no longer appropriate, and must be modified to be effectively applied to use in master/worker systems [96].

5.1 Rethinking Time Warp for Master/Worker PDES

A key consideration in the master/worker paradigm is client/server communication. This must be taken into consideration in all aspects of the optimistic execution. For example, in a conventional Time Warp system, a throttling mechanism is required to avoid spending an excessive amount of processor time on computations that will be later rolled back as well as to avoid potentially time consuming rollback operations; the cost of being overly conservative is largely idle processor time. In a master/worker system, pausing a computation may result in additional client/server communication because LPs may be returned to the server once they have been suspended. This cost associated with overly conservative execution must be factored into the throttling mechanism.

Further, direct client-to-client communication is not allowed, necessitating careful design of the messaging and rollback mechanisms. A straightforward implementation of the master/worker paradigm, as described earlier, can lead to excessive delays in handling straggler messages. The LP generating the straggler must first be returned to the server, and the LP receiving the straggler must also be returned to the server before the straggler message and subsequent rollback can be processed. These delays can lead to increased amounts of rolled back computation. The standard master/worker paradigm must be modified to address this issue. Further, traditional Time Warp mechanisms such as handling straggler messages and message cancellation through anti-messages, direct cancellation, or some other means do not directly translate into an optimistic master/worker system.

Transmitting LPs and their associate state and message histories between clients and servers can be problematic because of the large amount of data that must be sent. The overheads associated with such transmissions must be considered in developing a suitable state saving strategy.

In public-resource computing infrastructures such as desktop grids, client volatility is a paramount concern. Under this system, no assumptions can be made that a work unit lease will ever result in a valid return due to possible machine suspension, reboots, crashes, or general network connectivity disconnects. This behavior is a primary motivating factor for employing a master/worker paradigm and further underscores the need to revise conventional techniques for optimistic PDES.

PDES master/worker systems can take advantage of the inherent centralized control in a master/worker paradigm greatly simplifying global control tasks such as calculating GVT. As message servers process incoming data and bin them to the appropriate output queue, the minimum time stamped message per work unit is kept track of and updated to the master service. The master uses these values to calculate the global minimum time stamped message in the entire simulation. This value is then used to calculate the GVT of the system. Additionally, proactive fossil collection is possible as GVT values are updated on every work unit lease to a client.

5.2 Towards Optimizing Optimism in a Master/Worker PDES System

Implementation of an efficient optimistic PDES system in the master/worker paradigm requires addressing a variety of new issues. The following sections detail

concepts, protocols, and mechanisms specifically supporting optimistic PDES codes in a master/worker framework.

5.2.1 Client-side Caching

The state of an LP/work unit in an optimistic system will become very large because of the need to maintain state and message history information. Transmitting this information back and forth between client and server will be very time consuming and expensive in terms of computing and communication resources.

To address this issue, client-side caching can be used by storing recent states on the client instead of notifying and updating the state service every time the state of the work unit is changed. The client stores modified states for recovery purposes in a state history. The current GVT value is piggybacked on any rollback notification or successful work unit completion. The GVT value is used during garbage collection to prune the state history queue. State history is also reduced after a rollback notification for any states exceeding the rollback time. While client-side caching is useful for conservative master/worker PDES, it is particularly crucial in optimistic systems.

An update policy is required to specify when the server is updated with changes in the client cache, analogous to the write-back policy used in traditional cache memory systems. Its importance is to balance the amount of updates to the state server to avoid using an excessive amount of network bandwidth, but still maintain enough state to prevent large coast forwards for fault recovery in the case of client failures. In the current Aurora implementation, the frequency of state updates can be tuned through configuration parameters.

In general, a replacement policy is needed to determine which work units are evicted from the cache when additional space is needed for a new work unit. An alternative approach is to statically allocate memory resources in the client, and enforce a policy never to accept new work units unless the memory resources are already available in the client. This latter approach is used in the current implementation for optimistic synchronization.

5.2.2 Time Windows and Zero Lookahead

All optimistic systems require a throttling mechanism to limit the amount of optimistic execution. Failing to do so can lead to instabilities and an excessive amount of rolled back computation. This is also true in master/worker systems; however, the throttling mechanism in master/worker systems takes on an additional meaning. The work unit may be returned to the master when the LPs contained within the unit are suspended. While the cost of overly restricting optimistic execution in traditional Time Warp systems only results in lost concurrency and in extreme cases idle processors, in a master/worker system it may result in additional communication and performance degradations. This suggests that it may be advantageous to allow greater levels of optimism in master/worker systems compared to traditional optimistic PDES systems.

The Aurora system allows work units to execute past safe execution limits but not arbitrarily far into the simulated future. Time windows are used to constrain execution but in a more relaxed fashion than a conservative implementation. These execution windows are necessary to provide points to check-in progress of a work unit to the back-end system so that recovery from faults and client crashes are possible. A separate time window is defined for each work unit, in contrast to the global time windows used in [97].

In general, it may be advantageous to set the time windows to a larger value than that which would be used in a traditional Time Warp system in scenarios where the number of available clients far exceeds the number of work units that are available.

One of the advantages of supporting optimistic simulations is the ability to execute PDES codes with zero or near zero lookahead. However, this poses a significant challenge when the execution mechanism is based upon time windows of some calculated length. Without lookahead, there is no reliable metric to determine execution boundaries except for the specified simulation begin and end times.

In general, zero lookahead simulations pose a problem for Time Warp in that they can lead to scenarios of unending rollbacks. This topic has been treated in the literature [98], and techniques to avoid these problems have been developed, e.g., by extending the precision of timestamp values so no two events have identical timestamps. Here, we assume such techniques are utilized and do not treat this topic further.

One approach to the time window problem is to adaptively tune window lengths as the execution progresses forward. Similar techniques have been proposed in conventional Time Warp systems. Using a rollback history for each work unit, the proxy can modify the leased execution window to either increase optimism if there are very few rollbacks occurring or to reduce optimism if the windows are too large [99].

The first implemented mechanism is to dynamically adapt time windows during the lease phase of the work unit. For zero lookahead simulations, the initial lease windows are based on the throttling parameter specified by the simulation package. After this initial lease window, the Aurora optimistic time management system performs dynamic window adaptation to tune the lease windows as the simulation progresses.

Instead of using static time windows calculated from GVT and the simulation end time, the time management system takes into consideration the recent rollback histories, average past time window lease lengths, and standard deviations of rollbacks. Depending upon the throttling aggressiveness chosen, the time management system will adaptively reduce the window if the standard deviation of the recent rollbacks exceeds a threshold value. Conversely, if there have been relatively no rollbacks from the histories, the system will adaptively grow the windows according to the throttling choice. The adaptive mechanism calculates time windows as follows. First the recent rollback average delta (ρ) is calculated which computes the absolute difference between recent rollback times to the total average rollback history. Non-negative ρ values denote that rollbacks are occurring or the rollback lengths may be increasing. Negative ρ values mean that rollbacks are not occurring or the rollback lengths may be decreasing. Next the coefficient of variation on rollback (υ) is calculated which is the rollback standard deviation divided by the rollback mean (μ). Other components to the time window calculation are computed as shown in Table 9.

Table 9. Additional Adaptive Time Window Components

	Lookahead-Based	**No Lookahead**
Begin Time (β)	Previous window end time	GVT
Optimistic Window (ω)	MinETS + tunable slice of input LA to WU	Tunable slice of total simulation time

$$\tau = \begin{cases} \beta + \omega - (\mu\upsilon) & \text{if } \rho \geq 0 \\ \beta + \omega + (\omega\upsilon) & \text{otherwise} \end{cases} \qquad (5.1)$$

The time window (τ) is calculated by summing the components as shown in equation (5.1). If the recent rollback average delta shows rollbacks occurring or increasing, the algorithm attempts to reduce the window size by some percentage of the rollback mean and coefficient of variation. If no rollbacks are occurring or rollback lengths are decreasing, the algorithm attempts increase the window size or keep the window size the same if the coefficient variation is near zero. Additional throttling components can be added to tune the time window adaptation to more aggressively or conservatively change time window lengths as the simulation executes.

The second mechanism is to reactively change the length of the time window when a work unit completes and begins its finalization phase with the proxy. Instead of authorizing the work unit to immediately begin the update phase, the proxy will perform a rollback history lookup and calculate a rollback average for the returning work unit. If the work unit window exceeds recent time window lengths calculated from rollback averages, the proxy will send a prune message to both the client and message server that is hosting the client's messages. This prune message contains an earlier end time than what was leased to the client for that iteration. The client will prune any messages sent exceeding the new time window end time and restore the proper state for that time as well. The message server will adjust the current lease times for the work unit to prepare for message delivery. This active pruning mechanism can reduce the number of potential rollbacks if the proxy determines that the current optimism of a work unit exceeds that of the previous calculated time window.

5.2.3 Self-Induced Rollbacks

There is a special case for rollbacks that must be considered in a master/worker PDES system. When a client finalizes a work unit, there is a possibility of rolling back its own execution. During the binning of messages from a work unit, a message may cause a rollback on another work unit which may cancel a message that was sent to the finalizing work unit which has been processed. This causes the returning work unit to perform a rollback on itself. If the client rolls back and returns messages that it already returned during the first finalization, there is a possibility of duplicate messages. The solution to this problem is to mask messages which have been already updated to the message server.

5.2.4 Adaptive State Saving

While adaptive state saving has been utilized on other Time Warp systems, existing approaches must be modified for use in master/worker systems mainly due to the approach of leasing execution windows to clients instead of in a traditional Time Warp system where nodes process events without defined check-out and check-in times. After a work unit lease, the client operates autonomously from the back-end system until the simulation reaches the given end time of the execution window. A rollback control message may be received during the execution or during the finalization phase when data is transmitted to the back-end services. A history of the past rollback times are recorded by the client. From these values, mean and standard deviation values can be calculated based on the most recent rollback times. These values can be used to filter state saves as shown Figure 63.

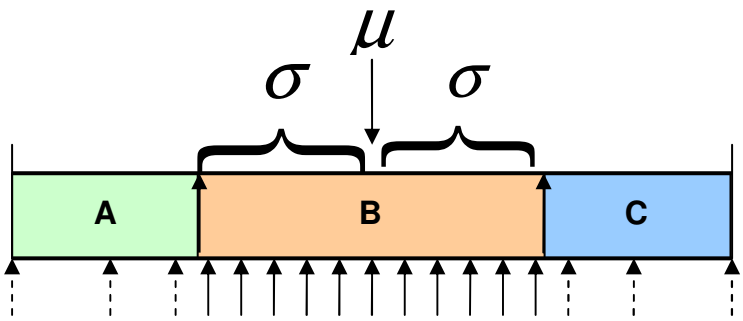

Figure 63: Computing Sub-Windows for Adaptive State Saves

From the calculated rollback average, the speculative execution sub-window shown as B in Figure 63 can be created. Within this window, state is saved regularly as there is an increased probability of a rollback occurring within this phase. There are two outlying sub-windows, A and C. The prior sub-window is referred to as the forward execution phase and the latter being the unreliable execution phase. The amount of state saving can be adaptively reduced during these windows. For the forward execution phase, state can be saved sparingly and then with increasing regularity as simulation time approaches the speculative execution phase. For the unreliable execution phase, the opposite is performed, as the simulation time moves away from the speculative execution phase, state is saved less frequently.

5.2.5 Unique Cancellation

Unlike conventional optimistic techniques, master/worker systems such as Aurora can detect rollbacks at the message server and correct errors on the server-side instead of at the workers or clients. When a rollback occurs, messages sent after the rollback time must be cancelled to correct errant execution. Conventional optimistic simulators often utilize anti-messages or direct cancellation through pointers. As discussed earlier, Aurora

exploits server-based message storage in the master/worker paradigm. Instead of generating anti-messages for each message that must be cancelled, Aurora utilizes delivery receipts and causal linkage information together with a 128-bit Universally Unique Identifier (UUID) of each message to *uniquely cancel* messages eliminating the need for any network overhead induced by anti-message generation if the message that must be cancelled resides on the same message server as the message causing the rollback. UUIDs are a method to uniquely identify data across network resources and can be generated without any centralized control. Additionally, UUIDs have an extremely low occurrence of collisions or probability that a same UUID will be generated over the course of the simulation program.

Using UUIDs for message cancellation is also important when the message service is distributed across multiple machines. If a rollback occurs on one message service, but the message does not reside on that server, the message service will perform a bulk unique cancellation where all cancellations are packed together by message service key and host address and sent as a single message. The receiving message server unpacks the bulk unique cancellation message and will then cancel messages locally. Note that this process may cause secondary rollbacks which are handled by the message service in a similar manner.

5.2.6 Delivery Receipts and Causal Linkages

In traditional Time Warp systems, if a message is received in the past, messages that were generated after the straggler message must be cancelled through some means such as anti-messages. In a master/worker system, the storage service for messages only needs two important pieces of information within each encoded message: the destination

work unit and timestamp of delivery. Upon finalization of a work unit, the message service would digest incoming messages and bin them into input queues of their respective work unit. For optimistic synchronization, however, this information is insufficient. Additional fields are required in each message sent by the PDES application that is then wrapped as an Aurora message. The simulation time at which the message was generated was incorporated to provide the necessary information for rollbacks and message cancellation. Moreover, each message is tagged with a UUID for identification of each message for cancellation purposes.

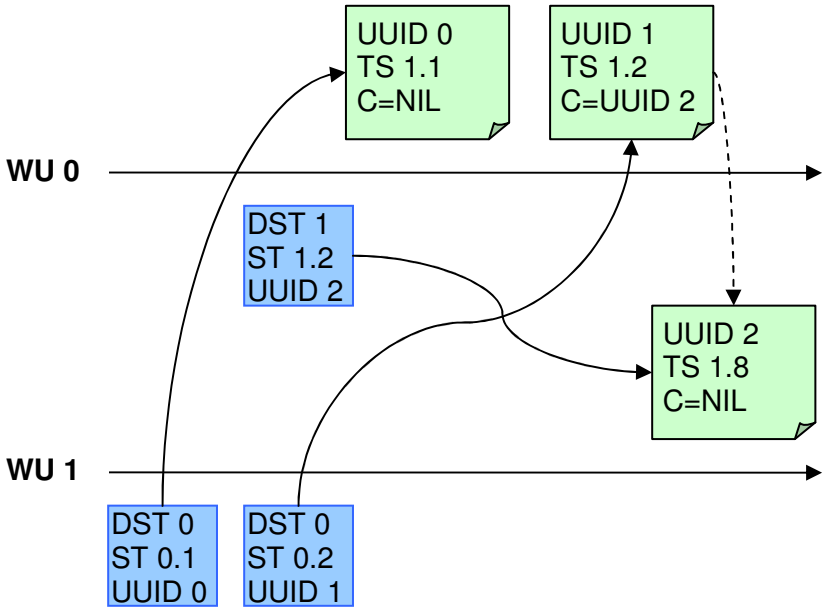

Figure 64: Causality Linkages and Delivery Receipts

When a message is generated by an LP, two additional objects are automatically created by the Aurora client. If a message is sent by the application and is generated due to an event being pulled from the input message queue and processed on the client, then a pointer (causal link) to the new message is created. These causal linkages are recorded

by the client linking a parent message with any children messages through the use of UUIDs.

The second object generated upon message send is the delivery receipt of the message. This specifically fulfills the need for delivery information in each message for the corresponding message server that only referenced destination work units and delivery timestamps, clearly insufficient for rollback and cancellation purposes. Since the client only works on one work unit at a time, any generated message is bound to the leased work unit and these delivery receipts can be returned to the same message service handling the input queue. Delivery receipts are sorted by the message server in increasing send timestamp order. When a rollback is detected, the message server simply references the delivery receipts and can identify messages that need to be cancelled without the inefficiencies of traversing priorities queues of each input queue.

In Figure 64 if a rollback were to occur on work unit 1 at time 0.15, the message service would look for delivery receipts with send times (ST) greater than or equal to 0.15. The second delivery receipt in the work unit 1 queue satisfies this requirement so it performs a lookup on the destination queue and associated message. When this message is cancelled, it also looks for any messages generated that were caused by that message. Since this message caused (C) the generation of a message with UUID of 2 to be delivered to work unit 1, this message is cancelled along with the message with UUID of 1.

5.3 Rollback Protocols for Master/Worker PDES

The master/worker paradigm calls for new types of rollback mechanisms to maximally conserve work and to minimize communication overheads. As in traditional Time Warp systems, rollbacks occur when a message is received in an LP's past. However, in the master/worker paradigm, messages that cause rollbacks can be detected at the server once the work unit is returned instead of at the client where events are processed. The LP being rolled back may be contained in a work unit that is being leased to another client. Thus, begs the question of where straggler messages should be detected and where the message unsending mechanism should occur.

The advantage of implementing the rollback mechanism on the client is that mechanisms are well understood and straightforward as this route is most similar to existing Time Warp rollback protocols. Anti-messages are generated for each message sent and state histories are stored via some protocol such as copy state saving. The foremost problem with this approach is the increased network usage in a system where bandwidth is a premium commodity. Anti-messages must be propagated from the client to the back-end message service and then back out to the client. Clearly, this increases the inefficiency of the simulation. Additionally, more bookkeeping and new protocols must be devised to handle a client that fails or disengages during the middle of processing a straggler message and generating anti-messages.

In contrast to client-side straggler detection and recovery, a server-side approach may alleviate performance issues, especially bandwidth usage. During the binning process, the message service can check for straggler messages. If rollbacks are needed, the proxy service is notified and messages are cancelled immediately on the server side.

Clients are notified if their windows must be rolled back, but they only perform state vector recovery and input message queue fix up. No anti-messages are necessary in this approach. Ultimately, this approach is more efficient considering the necessity of keeping network activity to a minimum. Consequently, this approach calls for a different rollback mechanism, as described next.

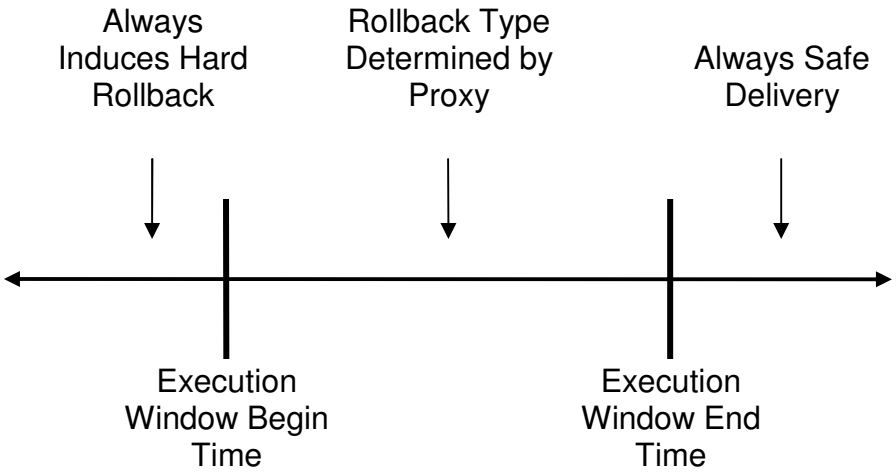

Figure 65: Message Delivery Cases

We introduce two types of rollback mechanisms specifically tailored for a master/worker paradigm and distribution of work using leased time windows. The type of rollback triggered is determined by the timestamp of a straggler message and is depicted in Figure 65.

If all messages are delivered after the leased execution window end time, then no rollback actions are necessary. If messages are delivered within the leased execution time window, then a rollback is triggered but the severity of the rollback is limited. If the proxy determines that the rollback time induces a "partial rollback," then this type of rollback is referred to as a *soft rollback*. A message that is delivered before the beginning

of the leased execution time window causes the proxy to direct the client to perform what is known as a *hard rollback*. Additionally, this condition can be triggered if a message is delivered within the execution window but the client has already completed finalization of the work unit.

5.3.1 Hard Rollback

A hard rollback is a rollback where the client cannot retain any previous computation and must restart its computation to a new simulation begin time set forth by the proxy. The client may be notified at anytime of a hard rollback, even during the simulation computation. The client can pre-empt the application runtime through the simulation loop control and restart the client with the new simulation parameters. A hard rollback uses a similar procedure as that given to a fresh work unit lease; the only difference is restoration of previous state information and the possibility of a coast-forward.

Once new lease parameters have been downloaded, the client will restore the most recent valid state from its internal state history. Any states that have been stored after this point can be safely discarded as they are no longer valid. The coast forward execution is similar to that in traditional Time Warp systems. In particular, during the coast forward phase message sending must be turned off. The client will contact the message server for messages with delivery timestamps falling inside the new execution time window. These messages will be inserted into the internal message queue. After these corrective measures have been processed, the client can proceed normally.

One of the features of Aurora is the ability of the application programmer to utilize the internal Aurora message queue as an event queue for the PDES application. In

this case, it is possible to perform "self-sends" where messages being generated by an LP are destined for itself. These messages must be properly re-inserted and re-processed during a hard rollback coast forward. Aurora automatically tags any self-sent messages and restores them, if necessary, to the input message queue for reprocessing if the delivery timestamp and message send time are greater than the specified rollback time.

5.3.2 Soft Rollback

Master/worker PDES systems could function properly using only the hard rollback mechanism alone. However, this would be inefficient as potentially good computation would be discarded for "partial rollbacks." Since messages can be delivered *within* a leased execution window, it is unnecessary to completely discard all computation a client has performed.

In contrast to the hard rollback mechanism, the soft rollback system allows the client to preserve the maximum amount of computation performed before the rollback notification is received. Once the soft rollback notification is received, the client will immediately set an internal flag that a potential soft rollback is about to occur. Before the client performs any further actions, a message cancellation list along with any associated new messages is downloaded from the message server. It is important to note that messages that have already been downloaded during the initial work unit lease are not downloaded again and that only new incoming messages are downloaded in bulk by the client.

If the client was pre-empted during application execution, it will check the current simulation time against the rollback time. If the rollback time occurs in the future, the client performs a *soft rollback intercept*. Instead of performing an actual rollback, the

client simply takes the new messages and inserts them into the input message queue. Any cancellations are processed as well. Once this task completes, the client is then resumed. This best case scenario preserves all useful computation.

If the rollback time is before the current client simulation time or the soft rollback notification happens during the finalization phase, a partial rollback must be performed. The proxy does not lease a new execution window. Instead, the window is renewed for the soft rollback. The client will compare the rollback time against the state history and will select the most recent history with a timestamp less than or equal to the rollback time. Similar to the hard rollback scenario, any states after this time may be safely discarded.

The first task during the soft rollback recovery phase on the client is to scan the message output queues for messages which were generated after the rollback time. These messages are pruned from the message queues and enqueued on the garbage collector memory list. Similarly, any causal linkage information or delivery receipts are erased. Next, the client will scan the original list of downloaded messages and re-insert any messages into the internal input queue which have a delivery timestamp after the rollback time and are not on the message cancellation list. Finally, the list of new messages associated with the soft rollback are inserted into the input message queue.

Utilizing this soft rollback mechanism allows full exploitation of any computation that has already occurred without rolling back the entire time window. The soft rollback is a new concept that was introduced to address issues arising in the master/worker paradigm.

5.3.3 Rollback Detection and Back-End Protocols

During the binning procedure for matching incoming message to their respective destination input queues, the message server will compare the delivery timestamp to that of the leased execution window. The message server will only make one distinction: whether or not the message will cause any kind of rollback. The message service does not need to differentiate between the two as the actual distinction will be determined at the proxy where the most up-to-date information on each work unit resides.

Messages that have a timestamp less than the last lease execution end time are tagged as potential soft rollback messages. These soft rollback messages are simply references to the actual message being placed into the input queues to avoid memory copy overheads. In addition to cataloguing these potential soft rollback messages, a mapping of rollback times to work units are also generated during the message binning process. The proxy need only be updated to the smallest rollback time per work unit.

Once all messages have been delivered "virtually," then the cancellation phase begins. For each rollback time catalogued for a work unit, delivery receipt send times are compared against that of the rollback time. If the send time exceeds that of the rollback time, then a cancellation must be generated for the sent message. The message service checks each cancelled message for causal linkages. If the message has causal links to child messages, then those messages must be cancelled as well. If the destination work unit queue exists on the same server, then the message service will perform a unique cancellation internally as detailed earlier. Messages that need to be cancelled that do not reside on the same message server are collated into a singular bulk unique cancellation message that is sent to the proper message server hosting the destination work unit.

After the message service finishes the rollback-cancellation procedure, the proxy is notified of the message server convergence. The proxy will then scan all rollback times and determine which rollbacks are soft or hard rollbacks. If a client must roll back and the time falls in between the leased execution window but the client is inactive, then a hard rollback notification is sent. In the case the client is active, a soft rollback notification is sent.

5.4 Performance Study

Like traditional time warp systems, performance of master/worker PDES systems depends largely upon the application. These systems perform well with models partitioned into work units that are computationally intense per lease with good computation to communication ratio. The purpose of this performance study is to show the impact of certain application characteristics on the Aurora system.

The Aurora system was compiled with gcc 4.1.2. Clients designated as *Xeon-A* are SMP machines with two Intel Xeon 3.2GHz processors and 6GB of memory. Each *Xeon-A* node runs RedHat Linux with a 64-bit GNU/Linux 2.6.9 kernel. Clients designated as *Xeon-B* are SMP machines with Intel Xeon 3.06GHz processors and 2GB of memory. *Xeon-B* nodes run RedHat Linux with a 32-bit GNU/Linux 2.6.18 kernel. Nodes are connected through Fast Ethernet links. To minimize external factors, only one instance of each Aurora back-end service was used and no optimizations from chapter 4 were enabled. The MIW default scheduling policy was used for all tests.

5.4.1 Lookahead Effects

For this test, the particle-in-cell hybrid shock discrete event simulation was used. The hybrid shock model was configured as 20 total cells, 400 initial ions per cell (IIPC), and a cell width of 0.00025. The simulation was partitioned into 10 work units over 10 *Xeon-B* clients.

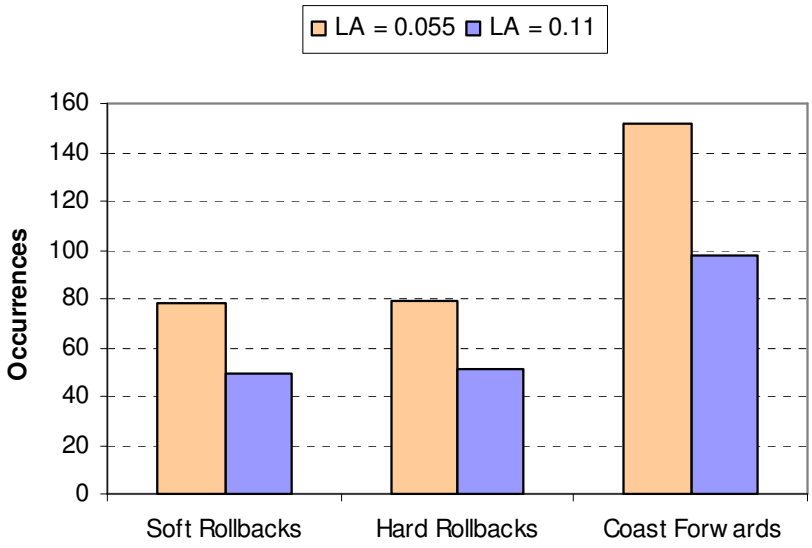

Figure 66: Effect of Lookahead on Rollback

In this test, the lookahead is set to a value of 0.11 and to half that value of 0.055. The maximum lookahead value that produced acceptable results was approximately 0.15. Figure 66 shows that smaller lookahead values lead to increased rollbacks in both the soft and hard rollback category. Similarly, the number of times the client must coast forward during a rollback is directly proportional to the number of rollbacks that occur. The current Aurora system does not have an upstream protocol to notify the back-end services that a message may potentially cause rollbacks on other work units as this would require a strict requirement of nodes to be connected for the entirety of the simulation, violating

one of the assumptions of client volatility. The time management system may also attempt to grow the execution window to further exploit any concurrency available to the model.

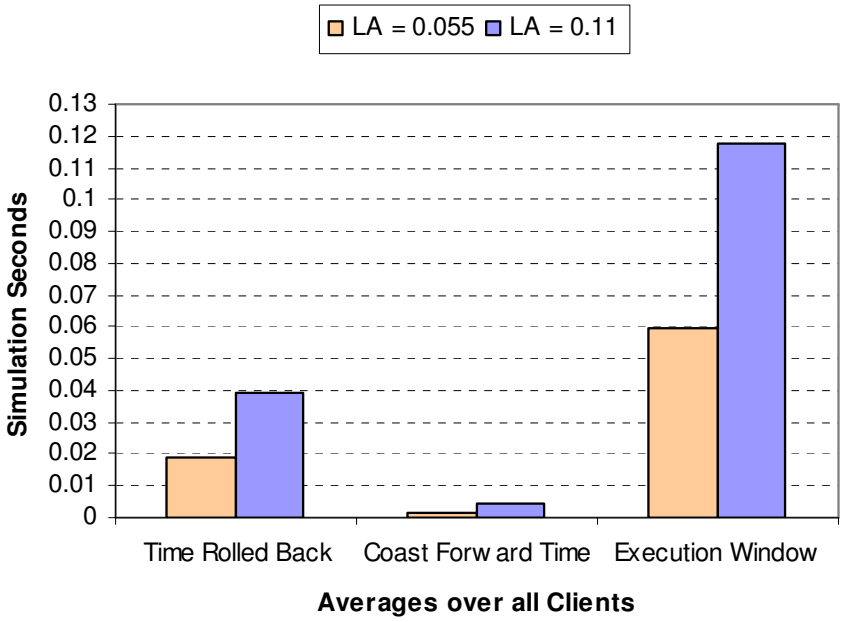

Figure 67: Lookahead Effects on Execution

With larger lookaheads, however, the simulation re-computation cost per rollback increases as shown in Figure 67. The amount of simulation time that must be rolled back and the coast forward time are both larger than that in the smaller lookahead model. The time management system also attempts to grow the execution window to further exploit any concurrency available to the model with execution window size gains of 8.3% and 6.9% for lookahead of 0.055 and 0.11, respectively.

5.4.2 Work Unit Granularity

The amount of useful computation given to each work unit lease is an important factor in determining performance. The hybrid shock simulation contains relatively large

amounts of state and messages and thus is not a particularly favorable simulation for master/worker PDES. However, the ability to scale the amount of computation per work unit lease allows for a good test case for work unit granularity. In this test, the hybrid shock parameters are the same from the previous scenario, except lookahead is kept constant at 0.11 and instead the IIPC parameter is modified. 10 clients on *Xeon-B* nodes were used.

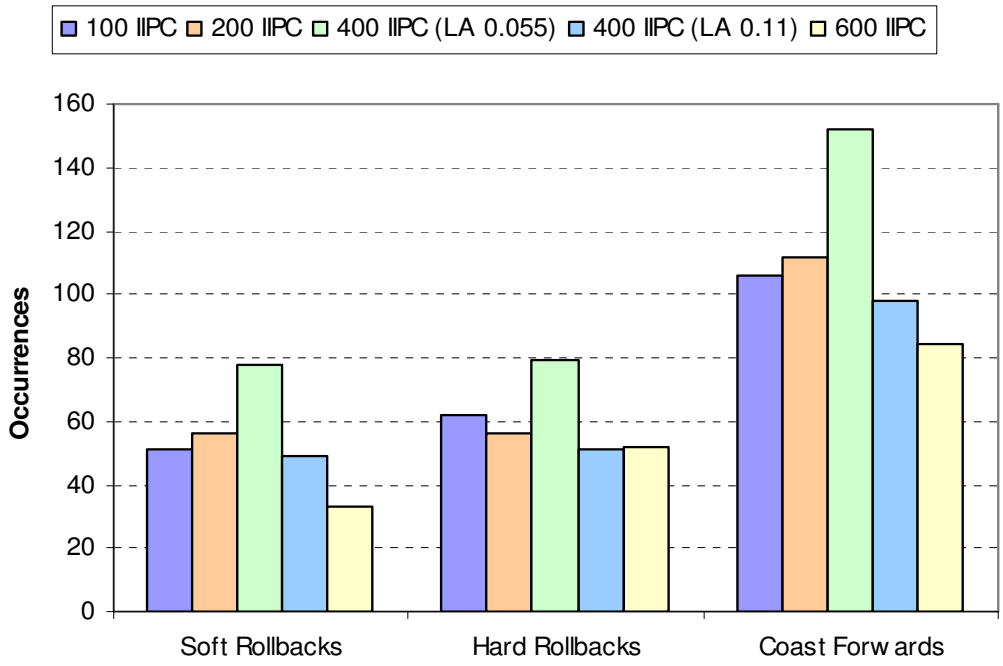

Figure 68: Hybrid Shock Rollback Trends

Figure 68 shows decreasing trends for both the number of rollbacks and coast forwards for the hybrid shock model as the relative amount of ions per work unit increases with the exception of the halved lookahead at 400 IIPC. As the amount of time spent in application code increases, the overall work unit return rate and possible rollback triggers decreases. Increasing the amount of work per work unit effectively constrains the simulation from advancing too quickly into the future lowering the amount of

overheads associated with rollback as clients spend more time in application code rather than attempting to advance further into the future with potential invalid computation.

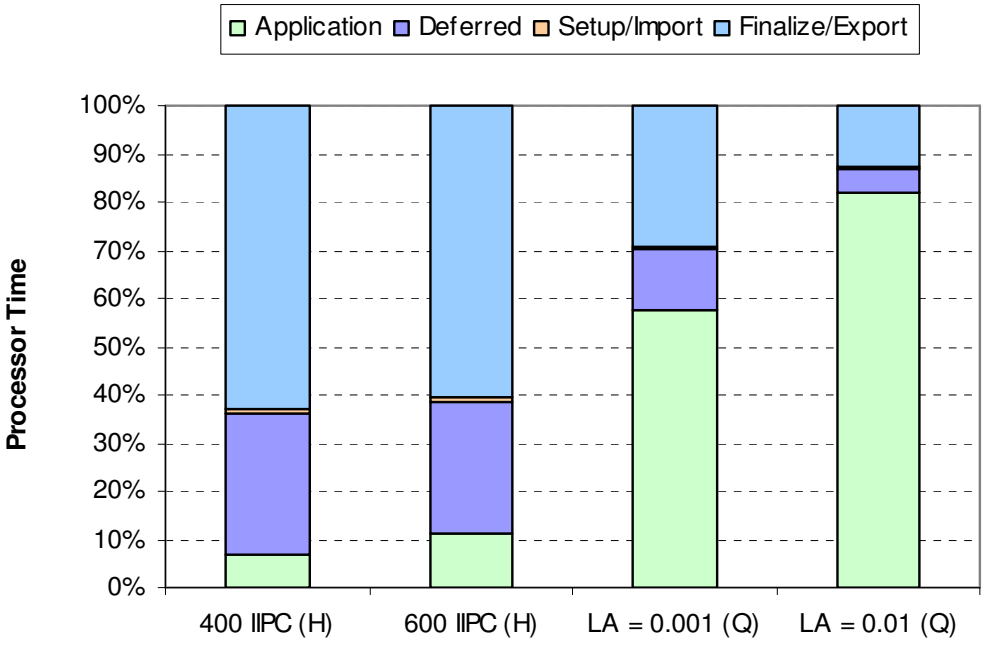

Figure 69: Simulation Efficiency

To test larger models in addition to hybrid shock, a closed torus queuing network simulation was used. This simulation model contains a 36x36 grid of queuing servers partitioned into 81 4x4 sub-grids for work units to lease. 1000 messages are generated with destinations local to the work unit and 1000 messages with remote destinations. The lookahead between work units is set at 0.01, and the service time is exponentially distributed with a mean of 5.0 time units. The queuing network simulation was run across 81 *Xeon-A* clients.

Figure 69 shows the amount of processor time dedicated to each phase of the client execution for the hybrid shock model (H) and the queuing network model (Q).

Application denotes processor time dedicated to actual simulation application computations, deferred refers to time spent by the client waiting for a renewed lease, setup/import time refers to the time spent in downloading lease metadata and input messages, and finalize/export is the amount of time spent in packing up generated messages and completing consistency convergence for the work unit return on the back-end.

For the hybrid shock simulation, the low amount of processor time dedicated to application code is due to the lack of ion interaction and memory limits of each client node. As the computational load increases per work unit, the relative amount of deferred wait overhead decreases as less stress is placed on the back-end services. The finalization phase consumes the most processor time as this includes overheads from triggering rollbacks since a work unit cannot proceed to a new lease until the consistency on the work unit has been validated by the back-end system and all new potential rollbacks have been acknowledged. Work is ongoing to improve performance by lowering communication and state saving overheads.

Increasing the lookahead in the queuing network model increases the work unit granularity as more events can be processed within a given lease window. Processor time spent in application code increases from 57.6% to 82.0% when increasing lookahead from 0.001 to 0.01. A non-linear increase in application processor time is due to the optimistic time management system finding additional concurrency in the 0.001 lookahead scenario; lease windows were on average 670% that of 0.001 lookahead permitting clients to run further into the future allowing events to be processed that may be safe from causality violations.

5.4.3 Adaptive State Saving

For analysis of the adaptive state saving mechanism, the hybrid shock simulation was utilized. The parameters for this model were 800 total cells, 100 initial ions per cell, lookahead of 0.11, and a cell width of 0.00025. The simulations were partitioned into 40 work units over 40 *Xeon-A* clients.

Table 10: Performance of Adaptive State Saving

	Disabled	Enabled	Difference
Average State History Size (MB)	535.58	402.90	-24.77%
Average State Save Time (wall sec)	5.46	4.11	-24.71%
Average Number of State Saves	10182.55	7936.57	-22.06%
Average Total Coast Forwards	10.50	11.62	+10.71%
Average Coast Forward Time (sim sec)	0.000470	0.000496	+5.71%
Average Client Runtime (wall sec)	62.642	61.893	-1.20%

Table 10 shows the benefits of enabling the adaptive state saving mechanism in Aurora. The amount of state and correspondingly the amount of processor time dedicated to state saving is reduced by approximately 25%. This can prove to be a significant gain for large scale models that must use optimistic synchronization allowing a greater portion of the memory available on the machine to be dedicated to the simulation instead of bookkeeping tasks concerning saved state. Over 2000 state saves are avoided at the cost of approximately 1 additional coast forward averaged across all clients. The additional

coast forward time is negligible compared to the savings gained from state save time and reduction in the total runtime.

Although these results are promising for adaptive state saving, further test scenarios are needed, especially those that examine zero lookahead applications as well as the tradeoff between increased coast forward probability and re-computation time for different PDES applications against more aggressive or relaxed state filtering algorithms.

5.5 Conclusions

Implementation of an optimistic time management system in Aurora enables execution of more PDES codes on public-resource computing infrastructures. Important properties such as small or zero lookahead are now supported in the Aurora framework. Given sufficient resources, the Aurora system can execute optimistically without regard for the output of any particular client as the results of an incorrectly computed work unit can be simply discarded.

Aurora exploits inherent advantages through the master/worker design, such as implementing straggler message detection and rollbacks on the back-end services without interaction from the clients. This reduces network bandwidth requirements and coupled with the unique cancellation mechanism provides direct cancellation of messages on work units hosted on the same service and bulk cancellation otherwise. Development of an optimistic master/worker system has led to the design of techniques such as soft rollbacks as well as addressing unique challenges to master/worker PDES in a loosely coupled distributed computing infrastructure.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

This thesis has investigated research issues regarding the fusion of PDES and master/worker systems across metacomputing environments. Master/worker infrastructures offer many inherent advantages such as system-level support for fault tolerance, semi-automated load balancing with true fine-grained client control, heterogeneous machine architecture and operating system support, and the ability to capture idle-cycles across a wide variety of resources that would otherwise be wasted. However, these benefits are not free. The cost is heavy impacts on overall system performance due to data transmission of state vectors and messages between workers and the master service on every work unit lease. These and other issues must be addressed to effectively bring a wide range of PDES codes into a true volatile metacomputing environment across a master/worker system. The primary contribution of this thesis is to explore these issues. Research contributions of this thesis are summarized as follows.

We have shown the viability of a widely portable approach to master/worker PDES through the use of web services. Such an approach gives maximum flexibility with regard to programming language agnostic implementations and true machine portability through web services standards such as XML, WSDL, and SOAP. We showed that certain PDES applications can perform well under this approach. However, we observed that large-scale PDES codes that contained large amounts of state vectors and messages performed poorly, exacerbated by the additional overheads introduced by

requirements imposed from web services such as XML encoding of all transmitted data. PDES applications that exhibit behavior such as large message sizes and traffic are not conducive to of the types of programs for which web services are targeted. Consequently, observation of performance data and consideration of limitations and issues surrounding master/worker PDES such as state and message serialization lead to a concurrent and distributed approach to master/worker PDES. Distribution of the master server into functionally different services combined with a slightly less portable communications framework using sockets and efficient binary message formats lead to a scalable architecture allowing dynamic back-end system scaling and support for large-scale PDES codes. We showed that by reducing the serializing effects (e.g., sequential operations such as import and export of data) of a strict master/worker system leads to significant reduction of overheads.

While the aforementioned optimizations provided significant performance and scalability gains with regard to the master/worker system architecture, true overhead reduction by way of reducing intrinsic overheads was addressed through optimization techniques to allow master/worker PDES codes to behave like traditional PDES applications. We showed through the use of work unit caching, pipelined state updates, pro-active message updating, and work unit scheduling policies exploiting PDES properties that the performance gap between master/worker and traditional systems can be closed significantly. Although it is understood that master/worker systems most likely will not reach the same performance potential offered by conventional systems, the robust execution abilities of a master/worker system offset and exceed what is offered through typical monolithic PDES frameworks.

This thesis addressed an important issue regarding what PDES applications are most suitable to this master/worker style of computation. We have characterized PDES codes based on their properties such as lookahead, granularity, computation to communication ratio, and size of exported data such state vectors and aggregate message sizes. These statements were verified through performance data showing that computationally intense PDES applications tend to perform best under master/worker with high efficiency ratings. Moreover, we introduce metrics more suitable for measuring performance of master/worker PDES such as overhead component breakdown and application processor time, that are more appropriate in the context of computational throughput rather than pure speedup that is traditionally used.

Finally, this thesis presents an approach to optimism on master/worker PDES. A case is made for new techniques that are developed to specifically address the unique challenges that are presented because of a master/worker system. Novel mechanisms such as hard and soft rollbacks, delivery receipts, and bulk unique cancellation combined with optimizations to existing techniques tailored for master/worker time window based execution such as adaptive state saving and adaptive time windows have addressed many of the research issues facing optimistic execution across master/worker PDES.

6.2 Future Directions

The fusion of an unconventional execution platform with PDES that has a rich history of research presents an extensive amount of additional opportunities for expanding what has been explored through the work presented in this thesis. The following are possible future directions with regard to master/worker PDES:

- **Adaptive work unit morphing.** A large factor impacting the performance of master/worker PDES systems is the granularity of the work unit. A work unit that is too small may not contain enough computation to merit the amount of metadata and state vector overhead that must be shipped back and forth between the server and client. A work unit that is too large may spend an excessive amount of time in transfer if the bandwidth is insufficient or may suffer from slow simulation progression if clients in the worker pool have insufficient memory to support the work unit size. Instead of placing the burden on the simulation modeler and developer, only the smallest work unit should be specified that can be equivalent to traditional logical processes in parallel simulations. The system will then, dynamically at run-time, perform "teaming" operations where work units are dynamically combined on the fly and shipped in the system as "super work units." This provides several benefits outside of reducing modeler complexity including: reduced communication overheads, direct work unit to work unit message sends within a super work unit, and automatic granularity tuning in response to the amount of bandwidth and processing power available in the system. Moreover, algorithms must be designed to evaluate decision points on when to break these "super work units" back into smaller pieces under situations such as network congestion or a change occurs to the composition of the worker pool.
- **Bandwidth congestion monitoring and network policy.** Issues dealing with bandwidth congestion and an enforceable network policy have not fully been explored. This can provide large benefits to adaptive protocols such as state and message updates. Sending additional data over the network when a link is already

congested may have overall negative impact on performance, increasing deferred wait and overhead times of other workers.

- **In-depth failure analysis and recovery protocol optimizations.** Additional studies where clients and back-end services are forcefully removed from a running simulation would provide more quantitative data on the performance of a master/worker PDES system under failure scenarios. Observed data and results from these tests can drive enhanced protocol design or new algorithms for minimizing performance degradation under these circumstances.
- **Integrated simulations with High Level Architecture (HLA) and the Test and Training Enabling Architecture (TENA).** Aurora provides an environment for simultaneous mixed-mode execution of parallel simulations, but currently there are no facilities for integrated inter-simulator communication. A master/worker paradigm can provide a tremendous amount of computational power and large-scale simulations spanning multiple functionally different applications can reap large gains from increased model fidelity or speedup. There are many issues that must be addressed such as distributed time management, inter-simulator messaging, and secure application insulation. Providing HLA or TENA support can increase the relevance of these master/worker simulation systems through enhancing compatibility of the Aurora system with a large existing simulation software base. There is an area of rich research where additional issues must be addressed such as integration of HLA time management and the absence of TENA time management along with object attribute propagation and publish-subscribe semantics.

- **Optimistic synchronization enhancements.** Optimistic master/worker simulations can be further enhanced by allowing multiple leases of each work unit with varying end times if the worker pool is sufficiently large. This allows "tiered" speculative execution where the best lease with the least amount of incorrect computation is committed. Additionally, an early rollback notification system can be implemented where message generation on a client notifies the proxy and can potentially stop incorrect computations thus reducing the problematic delay in messages inherent to master/worker. An adaptive finalization mechanism can be used to expand or prune-back the simulation execution window based on new information received after the lease was given to the client to reduce potential rollbacks or increase optimism over the original lease.
- **Improvement to eviction and replacement policies used in caching.** An eviction and replacement policy that includes active feedback from the master service can potentially increase cache hit ratio by targeting specific cache blocks that are known to have no effect on the system in the future, rather than using completely probabilistic eviction strategies.
- **Back-end load balancing.** Initial work unit distribution can be enhanced by measuring activity of each storage service in addition to memory load. This can help network congestion and services which are experiencing high processor load. Additionally, new protocols can be devised to perform dynamic runtime load balancing where state vectors and messages are migrated from high-load to low-load servers as simulations are running.

REFERENCES

[1] R. M. Fujimoto, *Parallel and Distribution Simulation Systems*. New York: John Wiley & Sons, Inc., 1999.

[2] A. Park, K. Perumalla, V. Protopopescu, M. Shankar, F. De-Nap, and B. Gorman, "On Evaluation Needs of Real-Life Sensor Network Deployments," in *Proceedings of the 2nd European Modeling and Simulation Symposium*, Barcelona, Spain, 2006.

[3] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley, "Large-scale network simulation: how big? how fast?," in *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, 2003, pp. 116-123.

[4] D. M. Nicol, "The cost of conservative synchronization in parallel discrete event simulations," *J. ACM,* vol. 40, pp. 304-333, 1993.

[5] J. S. Steinman, "SPEEDES: A multiple-synchronization environment for parallel discrete-event simulation," *International Journal in Computer Simulation,* vol. 2, pp. 251-286, 1992.

[6] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *Software Engineering, IEEE Transactions on,* vol. SE-5, pp. 440-452, 1979.

[7] R. E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," Massachusetts Institute of Technology1977.

[8] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM,* vol. 24, pp. 198-206, 1981.

[9] E. W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters,* vol. 11, pp. 1-4, 1980.

[10] D. M. Nicol, "Non-commital barrier synchronization," *Parallel Computing,* p. 549, 1995.

[11] R. Ayani, "A Parallel Simulation Scheme Based on the Distance Between Objects," in *Proceedings of the SCS Multiconference on Distributed Simulation*, 1989, pp. 113-118.

[12] B. D. Lubachevsky, "Efficient distributed event-driven simulations of multiple-loop networks," *Communicatons of the ACM,* vol. 32, pp. 111-123, 1989.

[13] D. R. Jefferson, "Virtual Time," *ACM Trans. Program. Lang. Syst.,* vol. 7, pp. 404-425, 1985.

[14] B. Samadi, "Distributed simulation, algorithms and performance analysis (load balancing, distributed processing)," University of California, Los Angeles, 1985, p. 240.

[15] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *J. Parallel Distrib. Comput.,* vol. 18, pp. 423-434, 1993.

[16] Y.-B. Lin, B. R. Preiss, W. M. Loucks, and E. D. Lazowska, "Selecting the checkpoint interval in time warp simulation," *SIGSIM Simulation Digest,* vol. 23, pp. 3-10, 1993.

[17] R. Rönngren, M. Liljenstam, R. Ayani, and J. Montagnat, "Transparent incremental state saving in time warp parallel discrete event simulation," *SIGSIM Simulation Digest,* vol. 26, pp. 70-77, 1996.

[18] D. West and K. Panesar, "Automatic incremental state saving," *SIGSIM Simulation Digest,* vol. 26, pp. 78-85, 1996.

[19] J. S. Steinman, "Incremental state saving in SPEEDES using C++," in *Proceedings of the 25th conference on Winter simulation* Los Angeles, California, United States: ACM, 1993.

[20] L. M. Sokol, D. P. Briscoe, and A. P. Wieland, "MTW: A strategy for scheduling discrete simulation events for concurrent simulation," in *Society for Computer Simulation*, 1988, pp. 34-42.

[21] P. M. Dickens and P. F. Reynolds, "SRADS with local rollback," in *In Proceedings of the SCS Multiconference on Distributed Simulation*, 1990, pp. 161--164.

[22] J. S. Steinman, "Breathing Time Warp," *SIGSIM Simulation Digest,* vol. 23, pp. 109-118, 1993.

[23] R. M. Fujimoto, "Time warp on a shared memory multiprocessor," *Transactions of the Society for Computer Simulation International,* vol. 6, pp. 211-239, 1989.

[24] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "GTW: a time warp system for shared memory multiprocessors," in *Proceedings of the 26th conference on Winter simulation* Orlando, Florida, United States: Society for Computer Simulation International, 1994.

[25] B. R. Preiss and W. M. Loucks, "Memory management techniques for Time Warp on a distributed memory machine," *SIGSIM Simulation Digest,* vol. 25, pp. 30-39, 1995.

[26] D. Jefferson, "Virtual time II: storage management in conservative and optimistic systems," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing* Quebec City, Quebec, Canada: ACM, 1990.

[27] S. R. Das and R. M. Fujimoto, "Adaptive memory management and optimism control in time warp," *ACM Transactions on Modeling and Computer Simulation,* vol. 7, pp. 239-271, 1997.

[28] Y.-B. Lin and B. R. Preiss, "Optimal memory management for time warp parallel simulation," *ACM Transactions on Modeling and Computer Simulation,* vol. 1, pp. 283-307, 1991.

[29] K. S. Panesar and R. M. Fujimoto, "Adaptive flow control in time warp," *SIGSIM Simulation Digest,* vol. 27, pp. 108-115, 1997.

[30] A. Ferscha, "Probabilistic Adaptive Direct Optimism Control in Time Warp," in *Proceedings of the ninth workshop on Parallel and distributed simulation* Lake Placid, New York, United States: IEEE Computer Society, 1995.

[31] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Transactions on Modeling and Computer Simulation,* vol. 9, pp. 224-253, 1999.

[32] "OpenMP," http://openmp.org/.

[33] "Optimize Managed Code For Multi-Core Machines," http://msdn.microsoft.com/en-us/magazine/cc163340.aspx.

[34] "MATLAB Parallel Computing Toolbox," http://www.mathworks.com/products/parallel-computing/.

[35] "The Great Internet Mersenne Prime Search," http://www.mersenne.org/.

[36] "distributed.net: General-Purpose Distributed Computing," http://distributed.net/.

[37] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM,* vol. 45, pp. 56-61, 2002.

[38] M. R. Shirts and V. S. Pande, "Screen Savers of the World, Unite!," *Science,* vol. 290, pp. 1903-1904, 2000.

[39] "Folding@home for Playstation 3," http://www.scei.co.jp/folding/en/.

[40] "World Community Grid," http://www.worldcommunitygrid.org/.

[41] "ClimatePrediction.net: Experiments in Climate Forecasting," http://climateprediction.net.

[42] K. Davis, A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, S. Pakin, and F. Petrini, "A Performance and Scalability Analysis of the BlueGene/L Architecture," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*: IEEE Computer Society, 2004.

[43] K. S. Perumalla, "Scaling time warp-based discrete event execution to 10^4 processors on a *Blue Gene* supercomputer," in *Proceedings of the 4th international conference on Computing frontiers* Ischia, Italy: ACM, 2007.

[44] D. Luebke, M. Harris, J. Krüger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, and A. Lefohn, "GPGPU: general purpose computation on graphics hardware," in *ACM SIGGRAPH 2004 Course Notes* Los Angeles, CA: ACM, 2004.

[45] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, and V. Pande, "N-Body simulation on GPUs," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing* Tampa, Florida: ACM, 2006.

[46] K. S. Perumalla, "Discrete-event Execution Alternatives on General Purpose Graphical Processing Units (GPGPUs)," in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*: IEEE Computer Society, 2006.

[47] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications,* vol. 15, pp. 200-222, 2001.

[48] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing,* vol. 5, pp. 237-246, 2002.

[49] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *IFIP International Conference on Network and Parallel Computing*, 2005, pp. 2-13.

[50] J. B. Fitzgibbons, R. M. Fujimoto, D. Fellig, S. D. Kleban, and A. J. Scholand, "IDSim: an extensible framework for Interoperable Distributed Simulation," in *Proceedings of the IEEE International Conference on Web Services*, San Diego, CA, 2004, pp. 532-539.

[51] K. Pan, S. J. Turner, C. Wentong, and L. Zengxiang, "A Service Oriented HLA RTI on the Grid," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*, 2007, pp. 984-992.

[52] Y. Xie, Y. M. Teo, W. Cai, and S. J. Turner, "Servicing Provisioning for HLA-Based Distributed Simulation on the Grid," in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation* Monterey, CA: IEEE Computer Society, 2005.

[53] W. Cai, S. J. Turner, and H. Zhao, "A load management system for running HLA-based distributed simulations over the grid," in *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, Fort Worth, TX, 2002, pp. 7-14.

[54] S. W. Reichenthal, "Re-introducing web-based simulation," in *Proceedings of the 34th conference on Winter simulation: exploring new frontiers* San Diego, California: Winter Simulation Conference, 2002.

[55] Y. Huang, X. Xiang, and G. Madey, "A Self Manageable Infrastructure for Supporting Web-based Simulations," in *Proceedings of the 37th annual symposium on Simulation* Arlington, VA: IEEE Computer Society, 2004.

[56] J. M. Pullen, R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse, and A. Tolk, "Using Web services to integrate heterogeneous simulations in a grid environment," *Future Gener. Comput. Syst.,* vol. 21, pp. 97-106, 2005.

[57] A. Cholkar and P. Koopman, "A widely deployable Web-based network simulation framework using CORBA IDL-based APIs," in *Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 2* Phoenix, Arizona, United States: ACM Press, 1999.

[58] A. D'Ambrogio and D. Gianni, "Using CORBA to Enhance HLA Interoperability in Distributed and Web-Based Simulation," in *Computer and Information Sciences*, 2004, pp. 696-705.

[59] K. A. Iskra, G. D. V. Albada, and P. M. A. Sloot, "Toward Grid-Aware Time Warp," *Simulation,* vol. 81, pp. 293-306, 2005.

[60] Z. Balaton, G. Gombas, P. Kacsuk, A. Kornafeld, J. Kovacs, A. C. Marosi, G. Vida, N. Podhorszki, and T. Kiss, "SZTAKI Desktop Grid: a Modular and Scalable Way of Building Large Computing Grids," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1-8.

[61] D. P. Anderson, "BOINC: a system for public-resource computing and storage," in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, PA, 2004, pp. 4-10.

[62] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: a generic global computing system," in *Proceedings of the first IEEE/ACM International*

Symposium on Cluster Computing and the Grid, Brisbane, Qld., 2001, pp. 582-587.

[63] T. M. Ong, T. M. Lim, B. S. Lee, and C. K. Yeo, "Unicorn: voluntary computing over Internet," *SIGOPS Oper. Syst. Rev.,* vol. 36, pp. 36-51, 2002.

[64] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra, "InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines," *Concurr. Comput. : Pract. Exper.,* vol. 16, pp. 449-459, 2004.

[65] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan, "Harmony: a desktop grid for delivering enterprise computations," 2003, pp. 25-33.

[66] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: a scalable lightweight architecture for high throughput computing," in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 19-25.

[67] D. Kramer and M. MacInnis, "Utilization of a Local Grid of Mac OS X-Based Computers using Xgrid," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*: IEEE Computer Society, 2004.

[68] D. P. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," in *Cluster Computing and the Grid (CCGrid)*, Singapore, 2006, pp. 73-80.

[69] "BOINC: Berkeley Open Infrastructure for Network Computing," http://boinc.berkeley.edu/.

[70] "Folding@home," http://folding.stanford.edu/.

[71] A. Acharya, G. Edjlali, and J. Saltz, "The utility of exploiting idle workstations for parallel computation," *ACM SIGMETRICS Performance Evaluation Review,* vol. 25, pp. 225-234, 1997.

[72] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova, and A. A. Chien, "Characterizing and evaluating desktop grids: an empirical study," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, 2004, p. 26.

[73] V. S. Sunderam, "PVM: a framework for parallel distributed computing," *Concurrency: Pract. Exper.,* vol. 2, pp. 315-339, 1990.

[74] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, CA, 1988, pp. 104-111.

[75] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing,* vol. 63, pp. 597-610, 2003.

[76] J.-P. Goux, J. Linderoth, and M. Yoder, "Metacomputing and the Master-Worker Paradigm," Mathematics and Computer Science Division, Argonne National Laboratory ANL/MCS-P792-0200, February 2000.

[77] Y.-B. Lin, "Parallel independent replicated simulation on a network of workstations," in *Proceedings of the eighth workshop on Parallel and distributed simulation* Edinburgh, Scotland, United Kingdom: ACM Press, 1994.

[78] D. Kondo, A. A. Chien, and H. Casanova, "Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*: IEEE Computer Society, 2004.

[79] L. F. Wilson and D. M. Nicol, "Experiments in automated load balancing," *SIGSIM Simulation Digest,* vol. 26, pp. 4-11, 1996.

[80] E. Deelman and B. K. Szymanski, "Dynamic load balancing in parallel discrete event simulation for spatially explicit problems," *SIGSIM Simulation Digest,* vol. 28, pp. 46-53, 1998.

[81] A. Boukerche and S. K. Das, "Dynamic load balancing strategies for conservative parallel simulations," *SIGSIM Simulation Digest,* vol. 27, pp. 20-28, 1997.

[82] P. Peschlow, T. Honecker, and P. Martini, "A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation," in *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*: IEEE Computer Society, 2007.

[83] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy, "Lightweight remote procedure call," *ACM Transactions on Computer Systems,* vol. 8, pp. 37-55, 1990.

[84] S. Chandrasekaran, G. Silver, J. A. Miller, J. Cardoso, and A. P. Sheth, "Web service technologies and their synergy with simulation," in *Proceedings of the 34th conference on Winter simulation: exploring new frontiers* San Diego, California: Winter Simulation Conference, 2002.

[85] A. Park and R. M. Fujimoto, "Aurora: An Approach to High Throughput Parallel Simulation," in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*: IEEE Computer Society, 2006.

[86] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing," in *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing*: IEEE Computer Society, 2002.

[87] R. A. van Engelen and K. A. Gallivan, "The gSOAP toolkit for Web services and peer-to-peer computing networks," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, 2002, pp. 117-124.

[88] R. A. van Engelen, "Pushing the SOAP Envelope with Web Services for Scientific Computing," in *Proceedings of the International Conference on Web Services*, Las Vegas, Nevada, 2003.

[89] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, and M. J. Lewis, "Toward characterizing the performance of SOAP toolkits," in *Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, PA, 2004, pp. 365-372.

[90] R. M. Fujimoto, "Performance of Time Warp under synthetic workloads," in *Proceedings of the SCS Multiconference on Distributed Simulation*, 1990, pp. 23-28.

[91] K. Perumalla, R. Fujimoto, and H. Karimabadi, "Scalable Simulation of Electromagnetic Hybrid Codes," in *International Conference on Computational Science*, University of Reading, UK, 2006, p. 8.

[92] "Portable Components (POCO) for C++," http://pocoproject.org/poco/info/.

[93] M. Makoto and N. Takuji, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.,* vol. 8, pp. 3-30, 1998.

[94] K. S. Perumalla, "μsik: A Micro-Kernel for Parallel/Distributed Simulation Systems," in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*: IEEE Computer Society, 2005.

[95] K. S. Perumalla, A. Park, R. M. Fujimoto, and G. F. Riley, "Scalable RTI-Based Parallel Simulation of Networks," in *Proceedings of the seventeenth workshop on Parallel and distributed simulation*: IEEE Computer Society, 2003.

[96] A. Park and R. Fujimoto, "Optimistic Parallel Simulation over Public Resource-Computing Infrastructures and Desktop Grids," in *12th IEEE International Symposium on Distributed Simulation and Real Time Applications*, Vancouver, BC, Canada, 2008.

[97] L. M. Sokol and B. K. Stucky, "MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm," in *Proceedings of the SCS Multiconference on Distributed Simulation*. vol. 22, 1990, pp. 169-173.

[98] V. Jha and R. Bagrodia, "Simultaneous events and lookahead in simulation protocols," *ACM Transactions on Modeling and Computer Simulation,* vol. 10, pp. 241-267, 2000.

[99] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher, "Parallel and Distributed Spatial Simulation of Chemical Reactions," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation* Rome, Italy: IEEE Computer Society, 2008.